NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

A NEURAL NETWORK APPROACH TO MULTISENSOR DATA FUSION FOR VESSEL TRAFFIC SERVICES

by

Leonard Phin-Liong Koh

March, 1995

Thesis Advisor: Murali Tummala

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE A NEURAL NETWORK APPROACH TO MULTISENSOR DATA FUSION FOR VESSEL TRAFFIC SERVICES		5. FUNDING NUMBERS
6. AUTHOR(S) Koh, Leonard, P.		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORIN G AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis explores the use of neural networks to perform multisensor data fusion for Vessel Traffic Services (VTS). It begins with a detailed study of the VTS system in order to identify the type of input data and other system features that are suitable for fusion. This is followed by a brief study of the various neural networks to evaluate their suitability for data fusion applications. The Kohonen's self-organizing feature map (SOFM) was identified as the most suitable neural network that can be used for data fusion, but it has some limitations that make it unsuitable for solving the VTS data fusion problem. A neural network data fusion model was proposed that consists of a modified SOFM and a double fusion resolver to solve the problem of double fusion in VTS. The proposed model is simulated in software and tested with measured input data supplied by the U.S. Coast Guard. Results of fusion tests indicate that the proposed fusion system performs well; thus, the proposed neural network fusion model has potential for implementation in the VTS system.

14. SUBJECT TERMS Neural networks, multisensor data fusion, vessel traffic services (VTS).			15. NUMBER OF PAGES 105
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

A NEURAL NETWORK APPROACH TO MULTISENSOR DATA FUSION FOR VESSEL TRAFFIC SERVICES

by

Leonard Phin-Liong Koh
Republic of Singapore
B.Eng.(Hons.), National University of Singapore, 1987

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1995

Author: Leonard Koh

Leonard Phin-Liong Koh

Approved by: Murali Tummala

Murali Tummala, Thesis Advisor

G. S. Gill

Gurnam S. Gill, Second Reader

M. A. Morgan

Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

This thesis explores the use of neural networks to perform multisensor data fusion for Vessel Traffic Services (VTS). It begins with a detailed study of the VTS system in order to identify the type of input data and other system features that are suitable for fusion. This is followed by a brief study of the various neural networks to evaluate their suitability for data fusion applications. The Kohonen's self-organizing feature map (SOFM) was identified as the most suitable neural network that can be used for data fusion, but it has some limitations that make it unsuitable for solving the VTS data fusion problem. A neural network data fusion model was proposed that consists of a modified SOFM and a double fusion resolver to solve the problem of double fusion in VTS. The proposed model is simulated in software and tested with measured input data supplied by the U.S. Coast Guard. Results of fusion tests indicate that the proposed fusion system performs well; thus, the proposed neural network fusion model has potential for implementation in the VTS system.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

This thesis investigates the application of neural networks to multisensor data fusion with emphasis on the U.S. Coast Guard's Vessel Traffic Service System (VTSS).

Multisensor data fusion deals with the problem of how to combine data from multiple (and possibly different) sensors in order to make inferences about a physical entity or situation that may not be possible or accurate with a single sensor alone. The importance of this technology can be seen from the fact that the DoD critical technology plan has identified data fusion as one of twenty critical technologies required to advance the U.S. military capabilities [1].

Neural networks are massively parallel distributed processing systems that have the ability to self learn and adapt to the environment. Neural networks are also able to work with weak assumptions about the underlying physical proces that produces the input data to the network. Neural networks try to mimic the human brain which has been known to perform computations in an entirely different way from conventional digital computers. Neural networks are known to perform much better than conventional computers in certain areas, such as pattern recognition [2].

Vessel Traffic Services (VTS) are provided by many major ports in the world to monitor and control vessel traffic in the vicinity of the harbour. It also enables the port authorities to carry out functions like search and rescue, law enforcement, pollution control, and marine safety. Vessel Traffic Service System (VTSS) are installed at Vessel Traffic Centers (VTC) to facilitate VTS. VTSS receives data from different sensors like radio, video camera, radar and satellite (see Chapter II for more details) to provide a complete picture of every vessel location in the harbour area. Existing VTS systems perform a limited amount of data integration manually, i.e., they rely on the human operator to perform most of the integration function. With computerization and upgrading of sensor electronics, automatic integration of sensor data becomes possible and will be required due to increase in projected

vessel traffic. This will relieve the human operator of the tedium and enables him to perform the more important tasks like making decisions when faced with a potentially dangerous traffic condition.

A. OBJECTIVE OF THE THESIS

The U.S. Coast Guard is currently upgrading the VTS facilities at a few major ports in the country and is also planning the inclusion of new technologies and capabilities in the vessel traffic service system. One such technology of interest to VTSS is multisensor data fusion based on neural networks. The literature for multisensor data fusion through neural networks is aimed at military applications [1]. These studies are experimental in nature and application specific, and none is suitable for VTS applications. The main objective of this thesis is to investigate the possibility of using neural networks to perform multisensor data fusion for VTS applications.

A detailed survey of the current and the past literature on topics related to neural networks and data fusion was conducted [3]-[8]. Next, different types of neural networks were compared and analyzed in order to identify networks that have potential for application to data fusion. The specific requirements of VTSS were used to make modifications to existing neural networks. To examine the behaviour of the proposed model, software simulation studies were performed. Real data sets supplied by the U.S. Coast Guard were also used to test the algorithm proposed here.

B. ORGANIZATION OF THE THESIS

This thesis consists of six chapters. Chapter II describes multisensor data fusion in detail, with emphasis on the major functional blocks of data fusion. It also introduces the components and structure of the particular vessel traffic service system that is used for this thesis research. Chapter III briefly introduces the main concepts and types of neural networks. It then identifies the type of neural network that is suitable for data fusion and

shows that a straightforward application of this network will not fuse the sensor data properly. The rest of the chapter then discusses a modified neural network model that is suitable for data fusion for the VTS system. Chapter IV describes the implementation issues such as simulation strategy, software design and algorithm implementation. Chapter V presents the results of the simulated system, and Chapter VI provides conclusions and recommendations for future research.

II. MULTISENSOR DATA FUSION

Multisensor data fusion is the process of combining data from multiple sensors in order to make inferences about a physical entity or situation. The sensors may be of the same type, e.g., multiple radars in a surveillance system, or of different type, e.g., IR sensors and radars in a tracking system [1]. Though multisensor data fusion is a relatively new field, human beings have been performing it all the time. The following example illustrates the essence of multisensor data fusion as performed by a human being: Suppose you are asked to turn to page 20 of this thesis. You will flip a few pages at a time using touch and vision (sensory data). The touch determines the number of pages to turn in one flip. The eyes verify the page number you have turned to. The brain compares the page number you expect to appear and the page number you have actually turned to and decides how many pages to turn in the next flip. Suppose that you are now blindfolded. With touch as the only sense, it will be difficult to reach the correct page. What you can only do is to start from the first page, count the page numbers as you turn one page at a time until a count of 20 is reached. This method is more prone to error. Suppose a second person reads the page number to you while you are still blindfolded, where now the hearing and the touch are the two senses. Thus, the additional sensory data (obtained through hearing) makes the task easier and more accurate.

From the preceding example, the following observations can be made. In multisensor data fusion, different types of sensors complement each other and in combination produce a better result. Typically, more sensors lead to better results. The human brain uses input from one sensor to make a prediction about some aspects of the observed data, and it uses another sensor to confirm that prediction. In this case, the brain uses multiple sensors to coordinate a task.

A. BENEFITS OF DATA FUSION

Waltz [9] describes many benefits of data fusion and their impact on operational advantages. The benefits can be both qualitative and quantitative. Qualitative benefits include improved operational performance, extended spatial coverage, extended temporal coverage, increased confidence (e.g., higher probability of correct inference), reduced ambiguity of inferences, improved detection, enhanced spatial resolution, and improved system reliability.

Quantitative benefits are usually in terms of improved accuracy of estimated location or identity of an entity [9]. This can only be computed based on the specific system used. For example, a typical radar has good accuracy in measuring the slant range but not the angular position. On the other hand, a forward-looking infrared sensor (FLIR) can measure angular position accurately but not the range of the target along the line of sight. Hence, we can obtain good estimate of the location of a target by using the slant range data from a radar and the angular position data from a FLIR than just using data from either sensor alone. The quantitative improvement depends on the performance of the specific sensors involved, environmental effects, and the specific algorithms used in the data fusion estimation process. Accurate computation of improvement requires simulations and covariance error analysis. More examples of quantitative benefits of data fusion can be found in Nahin and Pokoski [10].

B. A DATA FUSION MODEL

Many types of models can be used to represent the data fusion process: A functional model can show the functions, databases and interconnections; an architectural model is good for showing the hardware/software configuration, data flows and inernal/external interfaces. A mathematical model is suitable to describe the algorithms and logical processes. The most suitable way to explain the data fusion process here is to use the functional model based on Hall and Llinas [1].

Figure 1 illustrates a schematic of a functional model for the data fusion process [1]. The model receives input data from multiple sensors. Major functions of

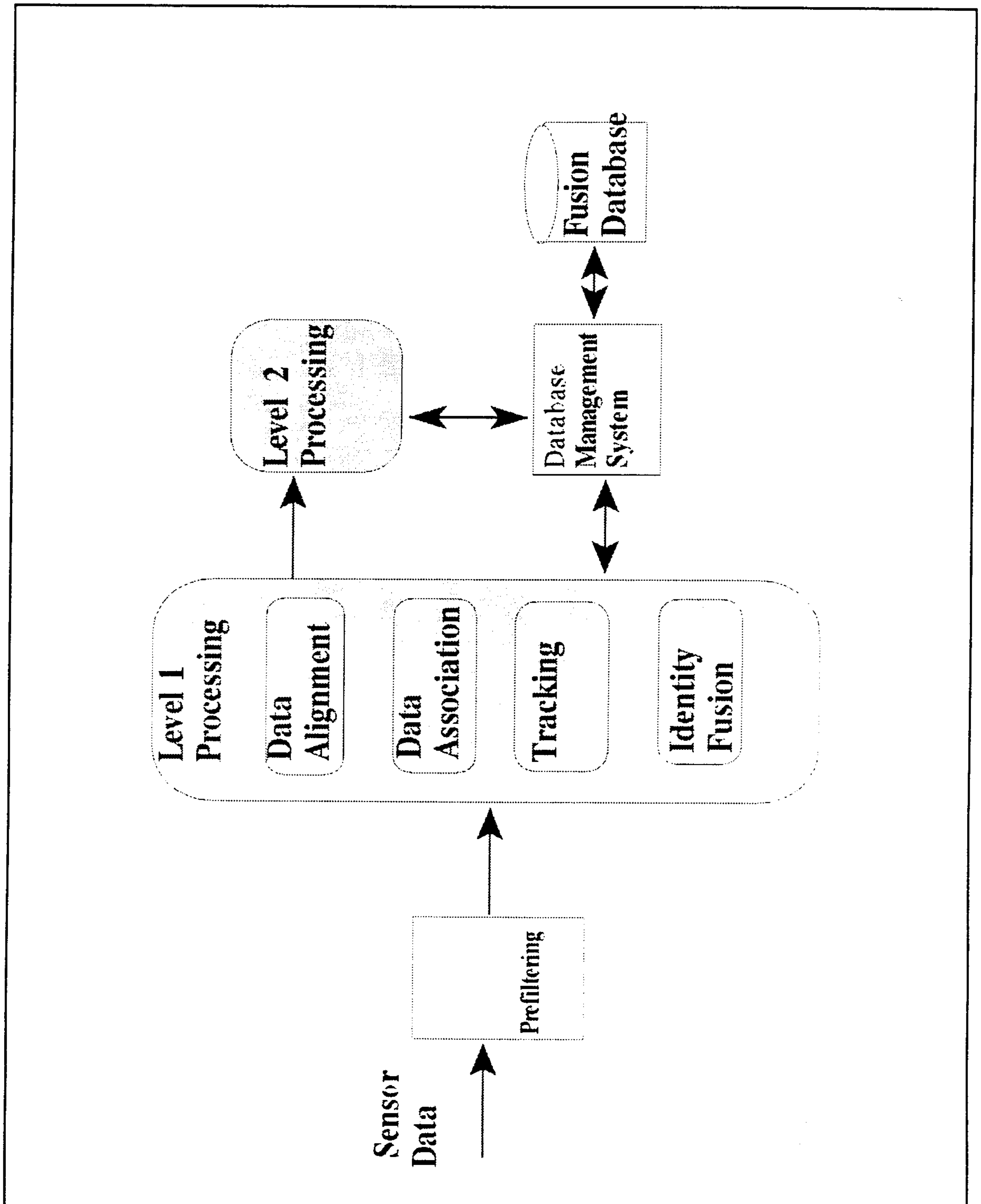

Figure 1. A Data Fusion Model. After Ref. [1].

fusion include prefiltering, level 1 processing and level 2 processing. Storing and retrieving of data is supported by a database management system.

Prefiltering of data helps to reduce the input data rate of the fusion system. If this is not done, the amount of sensor data from multiple sensors may be so large that it overwhelms the system. Prefiltering may be done by sorting data according to some common attributes, such as observation time, known locations or sensor type [1].

Two levels of processing are shown at the center of Figure 1. Level 1 processing fuses data to establish the position, velocity and identity of entities. It also establishes a database of identified entities and target tracks. Level 1 processing can be subdivided into four functions: data alignment, data association, tracking, and identity fusion [1].

Data alignment function transforms data received from multiple sensors into a common spatial and temporal reference frame [1]. This may include coordinate transformations (e.g., from latitude/longitude to x-y coordinates), time transformations (e.g., from individual sensor observation time to system time), and unit conversions.

Data association deals with the problem of sorting or correlating observations from multiple sensors into groups, with each group representing data related to a single distinct identity [1]. The computation is typically proportinal to N^2, where N is the current number of observations in the system.

Tracking refers to the process of using the obervations in a group to estimate the position and velocity of an entity [1]. Tracking is performed by updating the estimated parameter. It is easy to see that tracking algorithm is closely coupled to association schemes.

Identity fusion combines data related to identity (i.e., either names of entities or features that can be related to identity) [1]. Identity fusion techniques include clustering, artificial neural networks, template matching methods, Bayesian inference methods, Dempster-Shafer evidential reasoning, generalized evidence theory, and heuristic methods using expert systems.

Level 2 processing seeks a higher level of inference above level 1 processing.

The data are assessed with respect to the environment, relationships among entities, and patterns in time and space [1]. In the VTSS, for example, level 2 processing corresponds to determining if two vessels are on collision course.

C. VTSS AND DATA FUSION

A VTS is a service designed to improve the port safety by managing traffic within a port or waterway [11]. VTS is often compared to air traffic control due to their similarities in operational concepts. But due to historical reasons, VTS is not as standardized and established as air traffic control. The renewed interest in the development of advanced VTS is partly sparked by the Exxon Valdez disaster in March 1989 [12].

1. Overview of a Typical VTS System

An overview of the VTS system used by the U.S. Coast Guard is shown in Figure 2 [13]. The VTS comprises multiple Remove Site Subsystems (RSS) and a centralized Vessel Traffic Control Subsystem (VTCS). The RSS provides vessel tracking and communication sensors while the VTCS integrates, processes, stores and displays RSS data. The VTCS can be connected to more than 18 remote sites and up to 16 display consoles. Each RSS can acquire sensor data from up to 4 video cameras, 1 radar providing both digitized radar image video data and track data for up to 60 tracks, and audio for up to 2 transceivers and 4 guard receivers.

Additional sensor data containing a vessel's location and identity is received from differential Global Positioning System (DGPS) which is transmitted by each vessel to the RSS via a data link. DGPS is based on the knowledge of the accurate geographical location of a reference station, which is used to compute corrections to GPS parameters. These differential corrections are then transmitted to the GPS users, who apply the corrections to their received GPS signals or computed position [14]. Sensor data are multiplexed at each remote site for transmission to the VTCS via Coast Guard's terrestial data links.

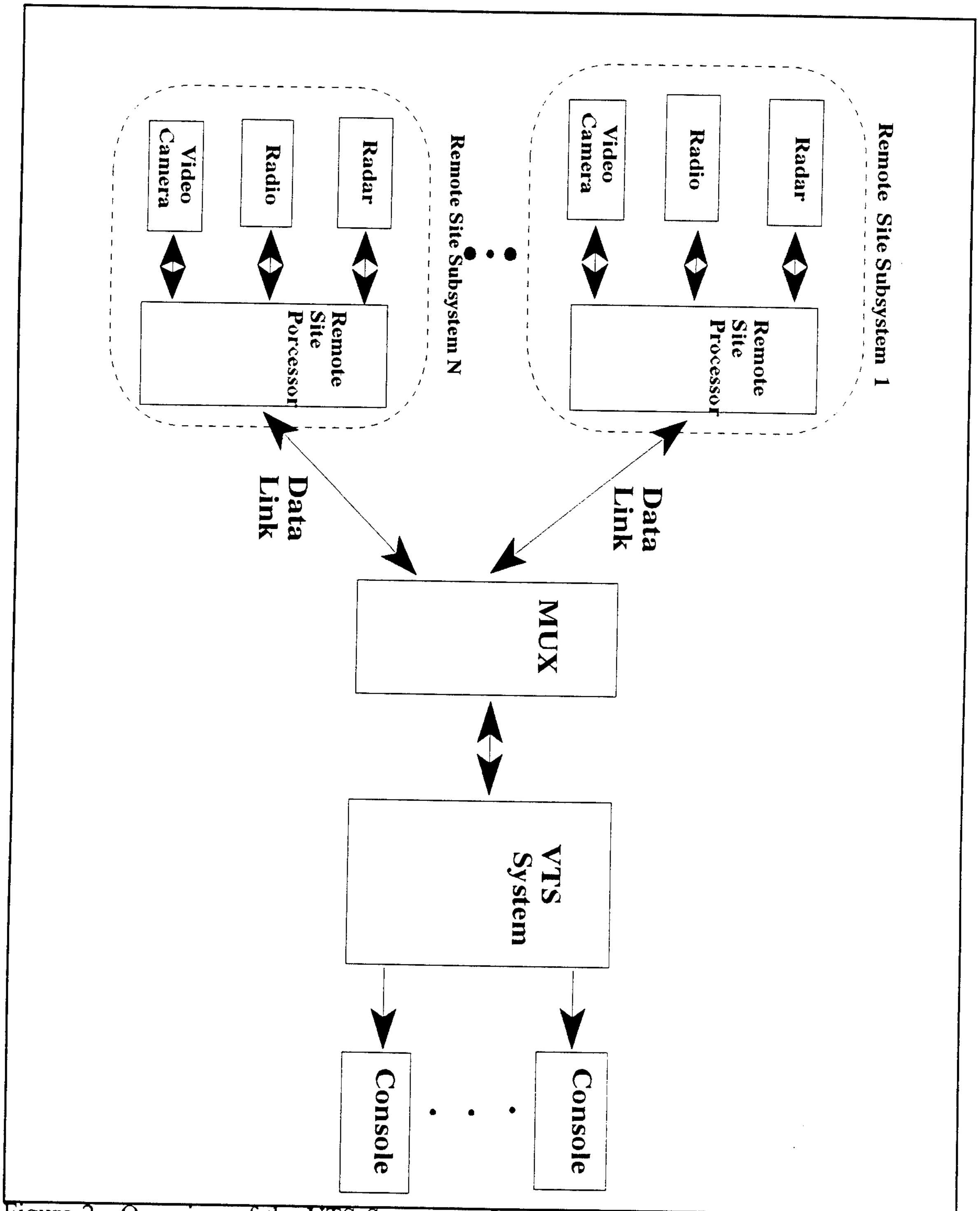

Figure 2. Overview of the VTS System. After Ref. [13].

2. Selecting Sensor Data for Fusion

To perform multisensor data fusion, the first step is to determine which sensor data provides information that is suitable for fusion. The following is an evaluation of the sensor data available to the VTS system.

a. Audio

Voice communication between the vessels and the VTC is provided via VHF transceivers. Though the audio signal can be digitized, information in the digitzed speech useful for fusion (e.g., vessel location, identity, heading) is expected to be only a very small part of the communication messages between vessels and VTS watch operator. As reliable automatic speech recognition systems are not available, digitized speech input to the fusion algorithm is not considered here.

b. Radar Track Data

At each remote site, there is a radar processor that can monitor 60 vessel tracks at any one time. The track data consists of time, latitude, longitude, speed, and heading. It should be noted that radars at remote sites have overlap regions of coverage which can be taken advantage of in multisensor data fusion to obtain more accurate results. Hence, the content and form of this data is suitable for fusion.

c. Digitized Radar Image Data

The digitized radar image data is actually the radar video that is processed by the remote site radar processor to obtain the track data. It is sent to VTC mainly for storage purpose and for possible later replay on the PPI display; hence, it is not considered for the fusion process. Nevertheless, this data does contain additional information that is not present in the track data.

d. Video

Cameras are mounted at remote sites to allow the VTC operators to have a direct view of the area of interest. But camera video may not provide useful information during bad weather and at night. Furthermore, automatic image recognition is difficult for this kind of application where vessels come in all sizes and shapes and appear at various distances and angles. The main use of the camera is for law enforcement and surveillance purposes. Hence, this data is not considered for fusion. Again, there is usable information in video data which could be explored in the future to achieve further fusion gains.

e. DGPS Data

With DGPS being made available to the public, it is expected that all vessels will be equipped with DGPS receivers in the future. Navigational accuraries better than 10 m [14] are achievable. Also, DGPS data contains information about the vessel which improves fusion performance.

III. NEURAL NETWORK FOR MULTISENSOR DATA FUSION

A. WHAT IS A NEURAL NETWORK ?

Neural networks, or Artificial Neural Networks (ANN), are massively parallel distributed processors that have the ability to learn and adapt to their environment [2]. Their strength also lies in the ability to work with weak assumptions or knowledge about the underlying physical processes that generate the input data to the network. Neural networks try to emulate the human brain by recognizing and processing patterns rather than crunching numbers. Though the idea of neural computing has existed for many years, it was the development of back-propagation and other important learning algorithms in the 80's that started the resurgence of interest in neural networks.

A typical neural network is shown in Figure 3. It consists of a layer of input nodes and a layer of output nodes, neurons and synapses. The shaded circles are known as neurons and the lines connecting the neurons to each other are called synapses. Each synapse is characterized by a weight.

The model of an individual neuron [2] is shown in Figure 4. A signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight, w_{kj}. The weighted sum, u_k, is offset by a threshold, θ_k, and passed through an activation function, $\varphi(\cdot)$, to obtain the output of neuron k, y_k. The operation of neuron k is defined by

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \tag{1}$$

and

$$y_k = \varphi(u_k - \theta_k). \tag{2}$$

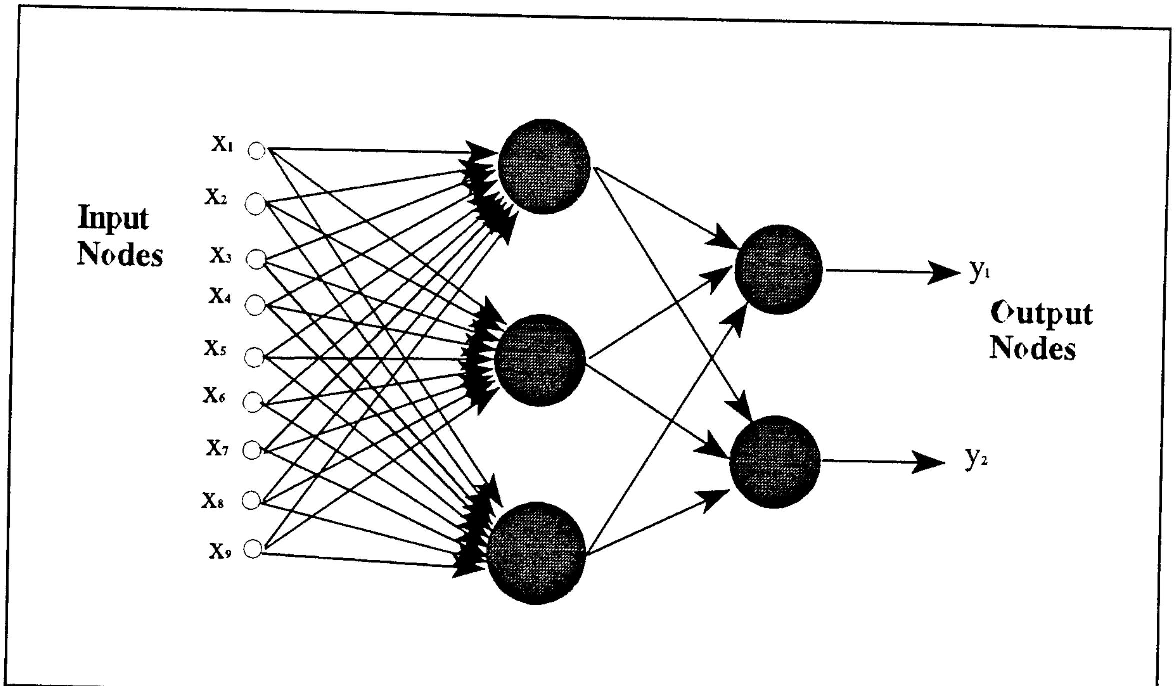

Figure 3. A Typical Neural Network.

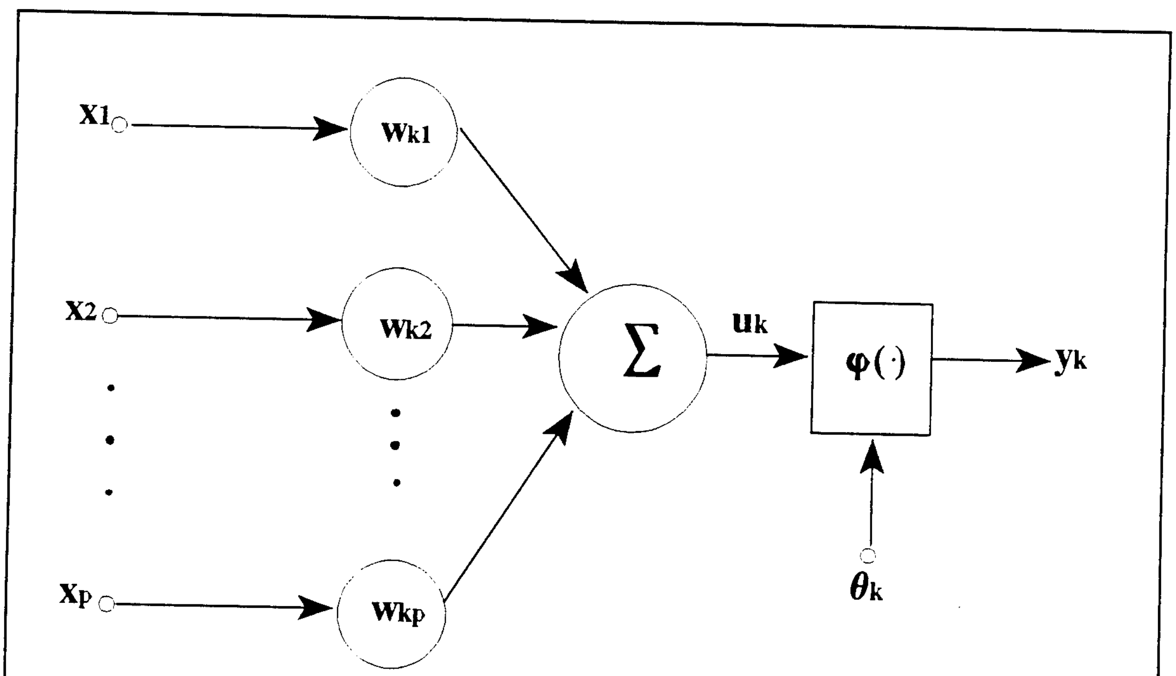

Figure 4. Model of a Neuron. After Ref. [2].

Common activation functions are the hard limiter, the piecewise-linear function, and the sigmoidal function.

B. CLASSIFICATION OF NEURAL NETWORKS

There are many ways to claasify neural networks. One way is to classify neural networks based on network architectures [2]. The first out of four network architectures is a single layer feedforward network. It consists of a layer of input (source) nodes and a layer of output nodes (neurons). An example is a linear associative memory. The second type of network architecture is multilayer feedforward netowrk. It is an extension of the single layer networks with the addition of one or more hidden layers of neurons. With more hidden layers, the network can extract higher order statistics especially when the size of the input layer is large. The third type is recurrent network, which is different from feedforward networks in that it has at least one feedback loop. The recurrent structure has a significant impact on the learning capacity of the network and on its performance. It can have hidden or no hidden neurons. The last type of network is the lattice structure. A lattice consists of one-dimensional, two-dimensional, or higher-dimensional array of neurons with a corresponding set of source nodes that supply the input signals to the array. A lattice network is really a feedforward network with the output neurons arranged in rows and columns.

The manner in which the neurons of a neural network are structured is closely linked with the learning algorithm used to train the network. Hence, it is also common to classify neural networks based on learning algorithms [2]. The common types of learning algorithms are error-correction learning, Hebbian learning, competitive learning, and Boltzmann learning. Error-correction learning is rooted in optimal filtering in that it tries to minimize a cost function based on an error signal. Hebbian learning is implemented using principal components analysis. The main idea is to perform eigenanalysis on a given pattern to extract the features. In competitive learning, the output neurons compete among themselves so that there is only one active

output neuron at any one time. Both Hebbian learning and competitive learning belong to a learning paradigm known as self-organized or unsupervised learning. This means that there is no "teacher" to tell the network whether the learning has been achieved correctly. The fourth is the Boltzmann learning. It is a stochastic learning algorithm based on information theory and thermodynamic theory. Error-correction learning and Boltzmann learning belong to the supervised learning paradigm, where a "teacher" is present to guide the network during the learning process.

C. NEURAL NETWORKS AND MULTISENSOR DATA FUSION

The first question that one would naturally ask is why we want to consider using neural networks for data fusion. The quick answer to that is that neural computing is a relatively new technology that has not yet matured. People are still discovering new applications for it. The interest in neural networks for data fusion is mainly due to their massive parallelism and ability to generalize. A short comparison between the use of neural networks and Bayesian statistical methods to perform data fusion is presented below [7].

The first difficulty encountered in using Bayesian methods is that the knowledge of conditional probabilities, a priori probabilities, probabilities of detection, etc., is required for computing decision estimates. In practice, such information is difficult or costly to obtain. Even if these parameters can be obtained, they have to be revalidated from time to time due to ageing of and upgrading/replacement of sensors. Mathematical modelling of the system is also required. This can be a complex task and may lead to poor performance due to overly simplified assumptions. Computations for Bayesian approaches can also be intensive if the amount of sensor data to be processed is large.

Neural networks, on the other hand, do not require any information about the statistical distribution of the sensors and the sensor observations. Due to their ability to learn, neural networks adapt to changes in system parameters. Because of this, there is no need to model the system mathematically. The massively parallel structure

of neural networks promises the potential of fast computing if the network is implemented in hardware.

1. Status of Technology

There is no known fusion model for the human brain. Fundamental research is still going on to understand how animals perform data fusion. So far, researchers have found six types of bimodal neurons in the optic tectum of the rattlesnakes [3]. These neurons integrate visible and thermal infrared sensory inputs. Ajjimarangsee and Huntsberger [4] have experimented with designing six fusion filters to emulate the rattlesnakes's neurons and using two other neural networks for pre- and post-processing. Pearson [5] have performed many interesting experiments to study how a barn owl integrates visual and acoustic signals to orientate its head to a target. The experiments include putting goggles and ear plugs on the barn owl to see how the lack of certain sensory inputs affect the accuracy with which the barn owl turns its head to face the target. Levine and Khuon [6] experimented with X-band radar, CO2 laser, and simulated passive IR spectrum as inputs to a fusion system to determine whether a missile has been launched or not. Identity fusion is used here. Multiple neural networks (multilayer perceptrons) are used as front end processors to estimate the identity, and the declared identities of all the networks are fed to another fusion network to determine the final identity declaration. Brown [7] applied neural networks to multi-source data fusion for passive airborne sensors. A fixed target is sensed by the same sensor at different times instead of being sensed by multiple sensors at the same time. Kagel [8] have built a three layer backpropagation neural network prototype in hardware to fuse multispectral imagery.

D. VTSS DATA FUSION USING NEURAL NETWORK

1. Proposed Neural Network Fusion Model for VTS

Figure 5 shows the proposed neural network data fusion model for VTS. A multiplexer (MUX) coordinates and manages the collection of sensor data from M number of radars and GPS/DGPS data from vessels. These sensor data are fed to the fusion neural network which performs the fusion function. Fused data are updated in the VTS database and retrieved as needed. A problem that is anticipated with the sensor data is called double fusion which will be explained in a later section. Neural networks cannot inherently handle the double fusion problem, so this problem is solved by a double fusion resolver.

2. Evaluation of Neural Networks for VTSS Data Fusion

None of the neural networks mentioned in the previous section is suitable for application in VTS. Most of them are tested under controlled laboratory conditions, and all of them fuse data that are statistically stationary. For VTS application, the sensor data are dynamic, i.e., a vessel's location, heading and speed change with time. This characteristic immediately rules out the use of multilayer perceptrons and Boltzmann machines. Both of them take a long time to reach convergence or stable state during the learning cycle. In fact, networks that require supervised learning are not suitable for VTS application. Hebbian learning, which uses principal components analysis, can be viewed as a feature extractor, which basically performs dimensionality reduction on the input patterns. Hence, its use has been mainly in the image compression area. But input data to the VTS data fusion system are already in a feature vector form. Each set of input vectors is uniquely related to a vessel, and the dimension of the input vector is small. Hence, a self-organized network that performs principal components analysis is not useful for VTS data fusion. That leaves two types of neural networks to consider: the Hopfield network and Kohonen's self-organizing feature map.

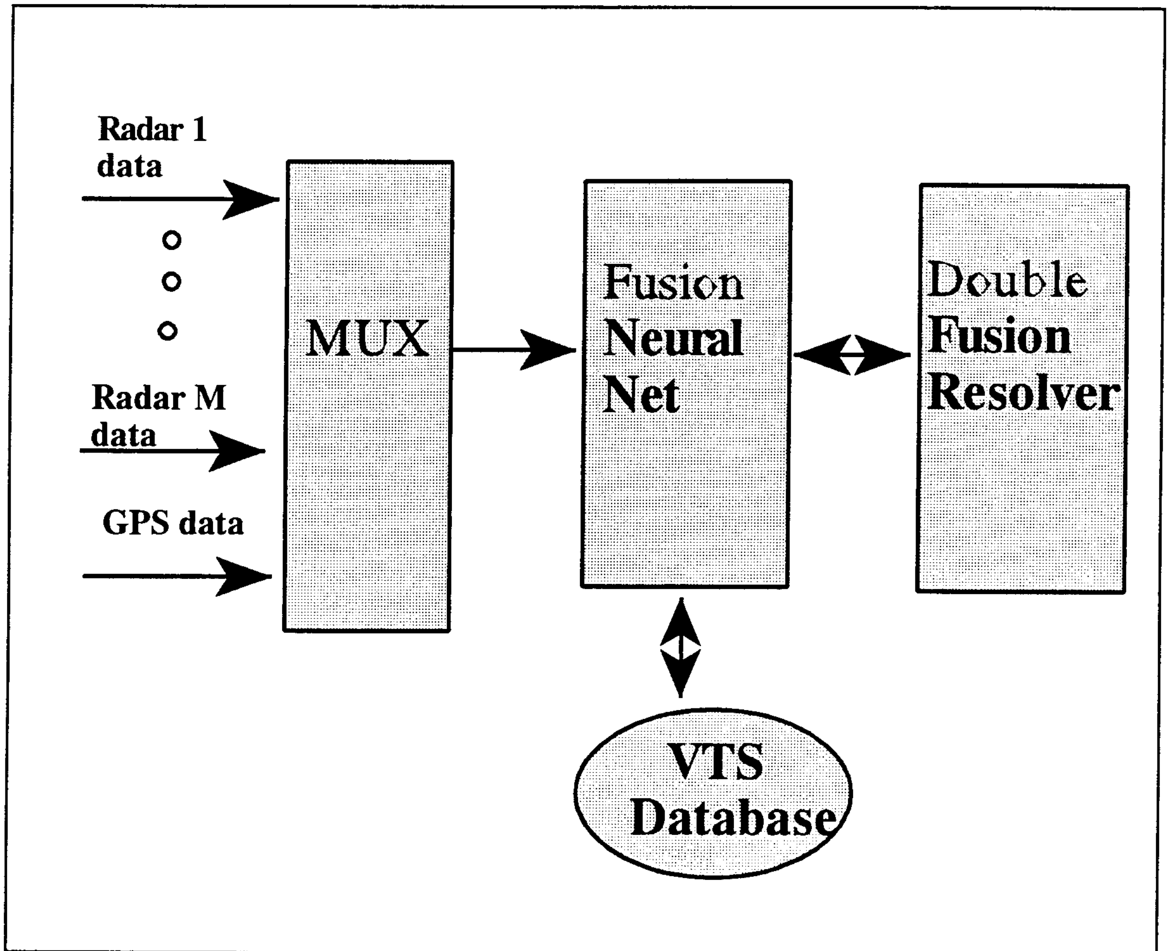

Figure 5. Proposed Neural Network Fusion Model for VTS System.

The Hopfield network has a recurrent structure. A typical Hopfield network is shown in Figure 6. Note that the output of each neuron in the network is fed back to all the other neurons, and there is no self-feedback (i.e., $w_{ii} = 0$). Because of this structure, the weight matrix of the network is symmetric. In matrix form, we have

$$W^T = W. \tag{3}$$

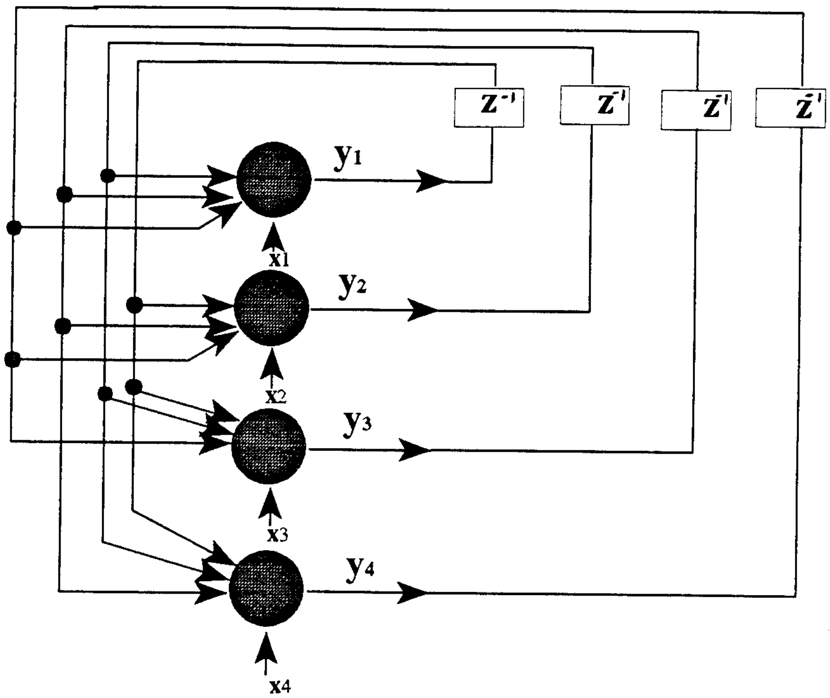

Figure 6. A Hopfield Network.

There is no learning phase for the Hopfield network. This characteristic makes it attractive for consideration in VTS application. But a closer examination reveals two serious limitations: convergence stability and storage capacity. There is no guarantee that the Hopfield network will converge to the right state. The state here can be taken to be the same as the reference input vector. This will be disastrous for VTSS. The maximum storage capacity, p_{max}, of a Hopfield network where all the fundamental memories are recalled perfectly is given by [2]

$$p_{\max} = \frac{N}{4\ln N}, \tag{4}$$

where N is the number of neurons. If N is 1000, the maximum storage capacity is only 36, ie., only 36 vessel tracks can be stored in the network. This is unacceptably low. This leaves us with Kohonen's self-organizing feature map which will be

examined next.

3. Kohonen's Self-Organizing Feature Map (SOFM)

Kohonen's Self-Organizing Feature Map (SOFM) [15] is a feedforward neural network with a single layer of neurons. It uses competitive learning. A schematic diagram of SOFM is shown in Figure 7. In competitive learning, all the neurons in SOFM compete to be the best matching or winning neuron. There is only one winning neuron for each competition. The weights of the winning neuron as well as those in the neighbourhood of the winning neuron are then updated. For example, if neuron 2 is the winning neuron in Figure 7, then neuron 1 and 3 are the neighbours of neuron 2 if the neighbourhood function is defined as the two closest neurons just next to the winning neuron. This process continues until convergence is reached. The neighbours of the winning neuron are defined according to a neighbourhood function, $\Lambda_{i(x)}(n)$. The general algorithm [2] is summarized below.

Let the input vector be denoted by

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_p]^T. \tag{5}$$

The synaptic weight vector of neuron j is denoted by

$$\boldsymbol{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jp}]^T, \qquad j = 1,2,\ldots,N. \tag{6}$$

The SOFM algorithm consists of the following steps.

a. Initialization

Choose random values for the initial weight vectors $w_j(0)$, j = 1, 2, ..., N.

b. Sampling

Draw a sample $\boldsymbol{x}$ from the input distribution; the vector $\boldsymbol{x}$ represents the input signal.

c. Similiarity Matching

Find the neuron whose weights are closest to the input vector $\boldsymbol{x}$. The comparison is done using a distance measure $d(\boldsymbol{x}(n), \boldsymbol{w}(n))$. The most common distance measure is the Euclidean distance. The neuron with the smallest distance is the winning neuron $i(\boldsymbol{x})$ at time n,

$$i(\boldsymbol{x}(n)) = arg\ \min_j d(\boldsymbol{x}(n), \boldsymbol{w}_j(n)), \qquad j = 1,2,...,N. \tag{7}$$

where *arg* means taking the argument of the expression on the right. This gives the index of a neuron. The output of SOFM is binary. The winning neuron has an output value of "1", and the other neurons have an output value of "0".

d. Weight Update

Adjust the synaptic weight vectors of all neurons using the update formula

$$\boldsymbol{w}_j(n+1) = \begin{cases} \boldsymbol{w}_j(n) + \eta(n)[\boldsymbol{x}(n) - \boldsymbol{w}_j(n)], & j \in \Lambda_{i(\boldsymbol{x}(n))}(n) \\ \boldsymbol{w}_j(n), & otherwise. \end{cases} \tag{8}$$

where $\eta(n)$ is the learning rate parameter, and $\Lambda_{i(\boldsymbol{x}(n))}(n)$ is the neighbourhood function centered around the winning neuron $i(\boldsymbol{x})$.

e. Repeat Step (b) to (d)

Iterate until no noticeable changes in the feature map are observed.

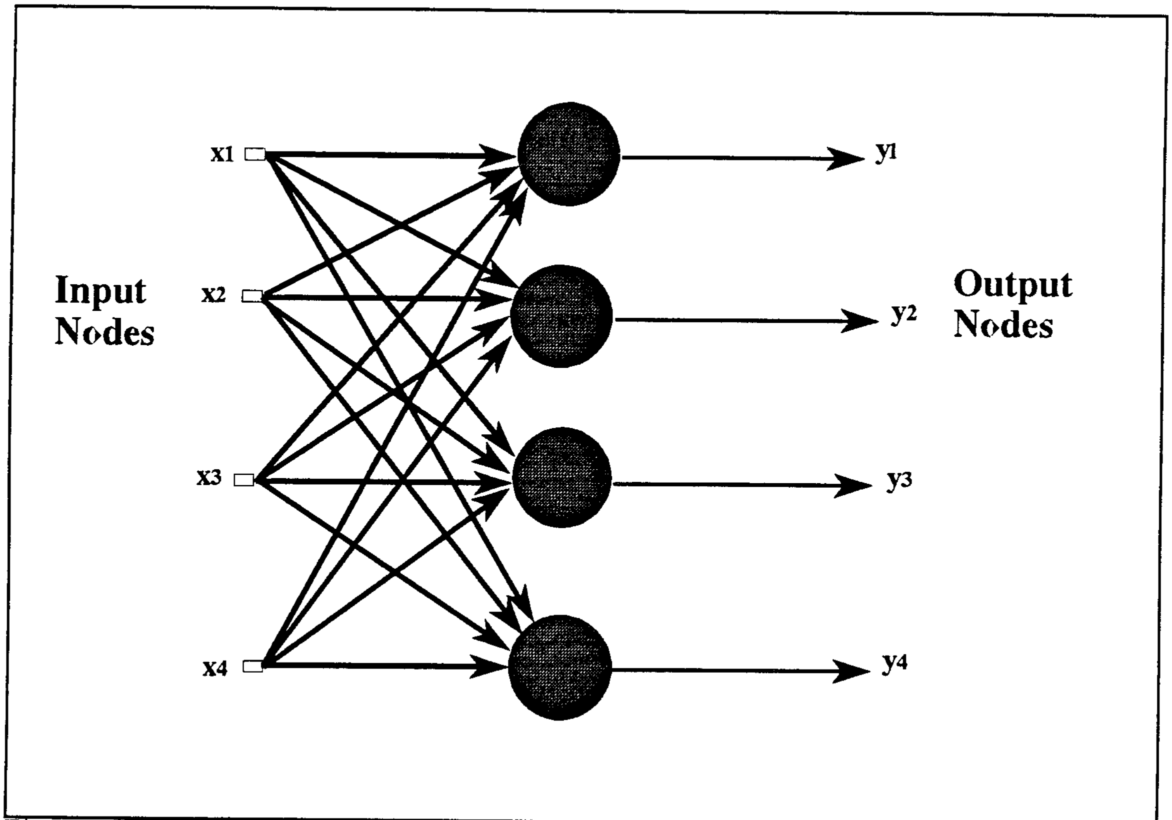

Figure 7. A Kohonen's Self-Organizing Feature Map.

E. A NEW NEURAL NETWORK MODEL FOR VTSS DATA FUSION

The standard SOFM cannot be applied directly to solve the VTS data fusion problem. This section discusses ways to overcome deficiencies of the standard network and proposes a modified SOFM model that is suitable for VTS data fusion application.

1. Winner-Takes-All Competition

In competitive learning, there is always one winning neuron from each competition. This works for applications where the class size is fixed and the input vector is always associated with one of the classes. For VTS application, it is possible that there is no winner in a competition. This happens when a new vessel has just been detected by the sensors. To overcome this problem, a threshold, θ, is used to

check the computed distance of the winning neuron. This can be expressed as

$$y_{i(x)} = \begin{cases} 1 & \text{if } d(\boldsymbol{x}, \boldsymbol{w}_{i(x)}) \leq \theta \\ 0 & \text{if } d(\boldsymbol{x}, \boldsymbol{w}_{i(x)}) > \theta \end{cases} \tag{9}$$

where $d(\boldsymbol{x}, \boldsymbol{w}_{i(x)})$ is the distance of the winning neuron from the given input vector $\boldsymbol{x}$. The choice of value for the threshold parameter will be discussed in Chapter V. Thus, this revised model can produce a null $\boldsymbol{y}$ vector if $\boldsymbol{x}$ corresponds to a newly detected vessel.

2. Fixed Class Size

The standard SOFM has a fixed number of neurons corresponding to a fixed class size. For VTS, the number of vessels being tracked at any one time is never constant. Some vessels may leave the port while others may arrive at any time. Since the number of neurons corresponds to the number of vessels being tracked, the algorithm should be able to dynamically adjust the total number of neurons. Specifically, when a new vessel is detected, there will be no winning neuron. The output vector will be null, and this condition prompts the algorithm to add one new neuron (or a previously used neuron which has been taken out of competition) to the network. Conversely, if a vessel leaves the port, the particular neuron representing the vessel needs to be disconnected from the network.

3. Weight Vector Initialization

The general SOFM algorithm initializes the weight vector to a random value. But when a new neuron is added to the network at time n, the input vector corresponding to the new vessel should be attracted to the new neuron. To make this happen, we initialize the weight vector of the new neuron j as feature vector of the new vessel.

$$\boldsymbol{w}_j(n) = \boldsymbol{x}. \tag{10}$$

4. Neighbourhood Function

In the general case, the SOFM algorithm uses a neighbourhood function, $\Lambda_{i(x(n))}(n)$, with a wide window around the winning neuron during the early phase of training and slowly shrinks the window with time until only the winning neuron is left. All neurons that are included in the neighbourhood function at time n are updated with the input vector $x(n)$. The main intention of the neighbourhood function is to address the neuron underutilization problem, i.e., some neurons never win a competition. But with the proposed model, only neurons that are needed are added to the network, and the unused neurons are removed from the network. So there is no neuron underutilization problem here. We may thus define the neighbourhood function to always cover only the winning neuron, i.e.,

$$\Lambda_{i(x(n))}(n) = i(x(n)). \tag{11}$$

This effectively means that the winning neuron has no neighbours.

5. Learning Rate Parameter

It is common to feed thousands of input patterns to the neural network during the training phase to reach convergence. But this will be unacceptably long for the VTS system which operates in real-time. Also, it should be noted that the features of a vessel are changing dynamically, so the network must give more emphasis to learning new features rather than past ones. To meet these requirements, learning is done during the time when there is no input to the network. To create learning patterns, the previous input feature vector is repeatedly applied to the input of the network The learning rate parameter, $\eta(n)$, is designed such that it decrease linearly over a fixed number of iterations from a high to a low value,

$$\eta(n) = \eta_i - \frac{(\eta_i - \eta_f)n}{L}, \qquad n = 0, 1, ..., L. \tag{12}$$

where η_i and η_f are the initial and final learning rate respectively, and L is the total number of learning iterations. The choice of actual parameter values are discussed in Chapter V.

For GPS/DGPS input data, the network should learn the information immediately since the reported location of a veessel is considered much more accurate than that reported by the radar sensors. The learning rate parameter is thus set to the maximum learning value of 1, which means that the winning neuron learns this information without any influence from the past values in its memory.

6. Distance Measure

The location of vessel computed by the sensor is in terms of latitude and longitude. The transformation, $\Phi(\cdot)$, is used to transform longitude and latitude, (u, v), to Cartesian coordinates, (x, y). To do so, a reference geographical location, (u_{ref}, v_{ref}), is selected as the origin (see section E in Chapter IV for the choice of this reference location). Thus, if $\phi = cos\ v_{ref}$, then [16]

$$(x, y) = \Phi(u, v) \tag{13}$$

with

$$\Phi_x(u) = (u - u_{ref})(111415.13\cos\phi - 94.55\cos 3\phi + 0.012\cos 5\phi) \tag{14}$$

and

$$\Phi_y(v) = (v - v_{ref})(111132.09 - 566.05\cos 2\phi + 1.2\cos 4\phi - 0.002\cos 6\phi). \tag{15}$$

Now $x(n)$ and $w_j(n)$ are separated in time as the vessel is generally in motion most of the time. Thus, the predicted current location of $w_j(n)$ has to be estimated using linear equation of motion denoted by $\Psi(\cdot)$. Thus, the distance measure is defined as

$$d(x, w_j) = \|\Phi(u_x, v_x) - \Psi(\Phi(u_{w_j}, v_{w_j}), c_{w_j}, s_{w_j}, t_{w_j}, t_x)\| \tag{16}$$

where

$$\Psi_x(x,c,s,t_1,t_2) = x + s(t_2 - t_1)\cos(\frac{\pi}{2} - c), \tag{17}$$

$$\Psi_y(y,c,s,t_1,t_2) = y + s(t_2 - t_1)\cos(\frac{\pi}{2} - c), \tag{18}$$

and $\|\cdot\|$ indicates Euclidean distance. This definition of distance measure allows us to directly measure the physical distance between an input vector and a weight vector in units of nautical miles or kilometres.

7. Double Fusion

No fusion algorithm can be guaranteed to resolve two closely spaced objects correctly. The problem is that two objects can be observed as one object if they are too close. In VTS system, radar tracks are sent from each radar as a buffer to VTC every 5 seconds. If two vessel tracks within a buffer are fused together, it is an errorneous condition known as double fusion. The proposed neural network fusion model has a mechanism to detect and correct double fusion. To detect it, the network marks each winning neuron when it is processing a radar buffer. If a neuron wins twice, double fusion has occurred.

The best way to explain the resolution of double fusion is through an example. Suppose the network has three neurons. Neuron 1 and 3 have won the competition on

the current buffer of data. If the next set of input data get fused to neuron 1, a double fusion occurs. This set of input data, together with the previous set of input data of neuron 1, is passed to a double fusion resolver. This resolver compares the two sets of input data with the weights of neuron 1 to see which set of data fits the weights best. The best-fit input data is retained by neuron 1 while the other set is open for competition among all neurons except neuron 1. If there is no winner in this competition, a new neuron is added to the network. Another possibility is that neuron 2 might win the competition in this round. Suppose neuron 3 won the competition, double fusion occurs again. The double fusion resolver is activated, and the cycle will go on until the new feature vector is either won by a neuron that has never won before or is assigned to a newly added neuron.

8. Weight Update Mode

Neural networks have two modes of updating weights. The most common one is called the pattern mode. In this mode, the weights are immediately updated after each presentation of the input pattern. The other mode is called the batch mode, where the weights are updated only after all the input patterns have been presented. Once the weights are updated, they cannot be restored to the values before the update. Hence, to be consistent with the handling of double fusion, the weights of winning neurons are not updated until the entire radar buffer has been processed. Thus the batch update mode is used for the VTS system.

IV. SIMULATION AND IMPLEMENTATION

In Chapter III, the proposed neural network fusion model was presented. This chapter presents the implementation of the fusion model. It begins with the discussion on the generation of sensor data, and describes the simulation of neural networks and the implementation of fusion database using Matlab. Extensive references are made to the program codes in Appendix B.

A. SOFTWARE DEVELOPMENT PLATFORM

The neural network fusion system was implemented in software. Matlab version 4.0 was chosen as the software development tool. Matlab's portability enables the developed program codes to run on either the SUN workstations or the IBM-compatible personal computers. It is important to bear in mind that the way the fusion system was implemented was strongly influenced by the specific software constructs of Matlab.

B. GENERATION OF SENSOR DATA

There are two types of sensor data: radar sensor data and GPS sensor data. Measured radar track data was provided by the U.S. Coast Guard while the GPS data was synthetically generated.

1. Radar Sensor Data

The U.S. Coast Guard has provided 24 sets of measured vessel track data which are listed in Appendix A. Each vessel track file contains the tracks of a vessel in ASCII format. To specify which vessels are involved in the simulation, the user enters a list of numbers corresponding to the numbers in the file names of the vessel track files. For example, if ship02 and ship14 are to be simulated, the simulation file will have numbers 2 and 14 listed in it.

The track entries in the vessel track files have a sampling interval of approximately 4 minutes. In the new VTS system, radar track data will be updated at approximately 5 second intervals. To achieve this faster update rate, data points between track entries have to be interpolated by re-sampling the track entries at a faster sampling interval of 5 seconds. As ships do not move in straight paths, the interpolated track entries were smoothed by a 9-point FIR filter (see cnysmo.m in Appendix B).

After generating the interpolated track entries, the next step is to simulate radar track data buffers. A radar track data buffer is simulated by grouping vessels with track entries that have the same time of observation.

2. Simulation of GPS Sensor Data

The GPS data provides the vessel position information with better accuracy than the radar track data. But there was no real GPS data available for testing. GPS data is provided infrequently and asynchronous in nature. To simulate this, a probability of GPS data arrival, P_{GPS}, is attached to each vessel. A number in the range of 0 to 1 is obtained from a random number generator with uniform distribution at each simulation iteration. If this random number is greater than P_{GPS}, the radar track data is marked as GPS data. If the number is less than P_{GPS}, no change is made to the radar track data. The value of 0.7 was chosen for P_{GPS}, for simulation (see vtss.m in Appendix B).

C. SIMULATION OF NEURAL NETWORK

The single layer SOFM is implemented with W representing the weight matrix. Each row of W is associated with a neuron. For example, the matrix element $W(2,1)$ is equivalent to weight w_{21}.

Two basic functions of the fusion neural network, i.e., adding new neurons to the neural network and training the network, are implemented (see Appendix B).

Add_new.m is a program that adds a new neuron to the neural network. This is implemented by either adding a new row to the W matrix or by reusing a row that has previously been deleted to indicate the removal of a neuron. Train.m is a program that puts the neural network in the training phase. During this phase, the weights of the winning neuron are updated. Fusion.m implements the competition among neurons in the neural network. As Matlab does not allow parallel processing, the competition is implemented sequentially. Distance measure computation is performed by fusedist.m, and double fusion resolution is performed by finefuse.m. To resolve double fusion, the winning neuron (that wins twice) retains the best matching track entry and rejects the other. To prevent the rejected track entry from being won by the same neuron again in the next competition, the computed distance of that neuron is artificially inflated to a large value.

D. IMPLEMENTATION OF FUSION DATABASE

The fusion database stores information such as vessel tracks, vessel arrival and departure schedules, physical characteristics of vessels, and other administrative information. Testing of the fusion algorithm does not require most of the database information. What are needed are the vessel identities and tracks. Specific fields of the database are the vessel names, the time the vessel was first detected, the time it was last updated, the assigned vessel I.D., a flag indicating whether GPS data was received, and the history of vessel tracks. Vessel tracks are stored in three matrices: trk_long, trk_lat, and trk_idx. This is because Matlab does not support matrices with more than 2 dimensions. Three database operations are supported: init_trk.m initializes the database, add_trk.m creates a new track for a newly detected vessel, and storetrk.m updates the tracks of a specified vessel (see Appendix B). The rest of the database information is stored in the additional columns of W for ease of reference. The database format is defined in dataform.m.

E. PRESENTATION OF FUSED DATA

Vessel tracks are plotted as they are being fused. A typical plot for two vessel tracks is shown in Figure 8. The seven radar remote sites are shown on the same plot (denoted by small circle) for reference. The seven sites are Brooklyn Naval Yard (BNY), Bank Street (BS), Governor's Island (GI), Mariners Harbour (MH), and Sandy Hook (SH). The geographical point with longitude 74°10' W and latitude 40°25' N is chosen as the origin of the x-y plane as it allows the port area to be covered. Two other locations are marked on the plot as additional reference points. The "current" time is shown on the top right hand corner. As a vessel is detected for the first time, a track number is assigned to that vessel. The track numbers and the names of the vessels are shown on the right hand side of the plot. Each vessel track is represented by a different color and line type (such as dotted or dashed lines)on the plot. plottype.m is the function that determines the color and line type to be used for each vessel track. The symbol "X" on the plot indicates the current position of a vessel. The main plotting program is plottrk.m (see Appendix B).

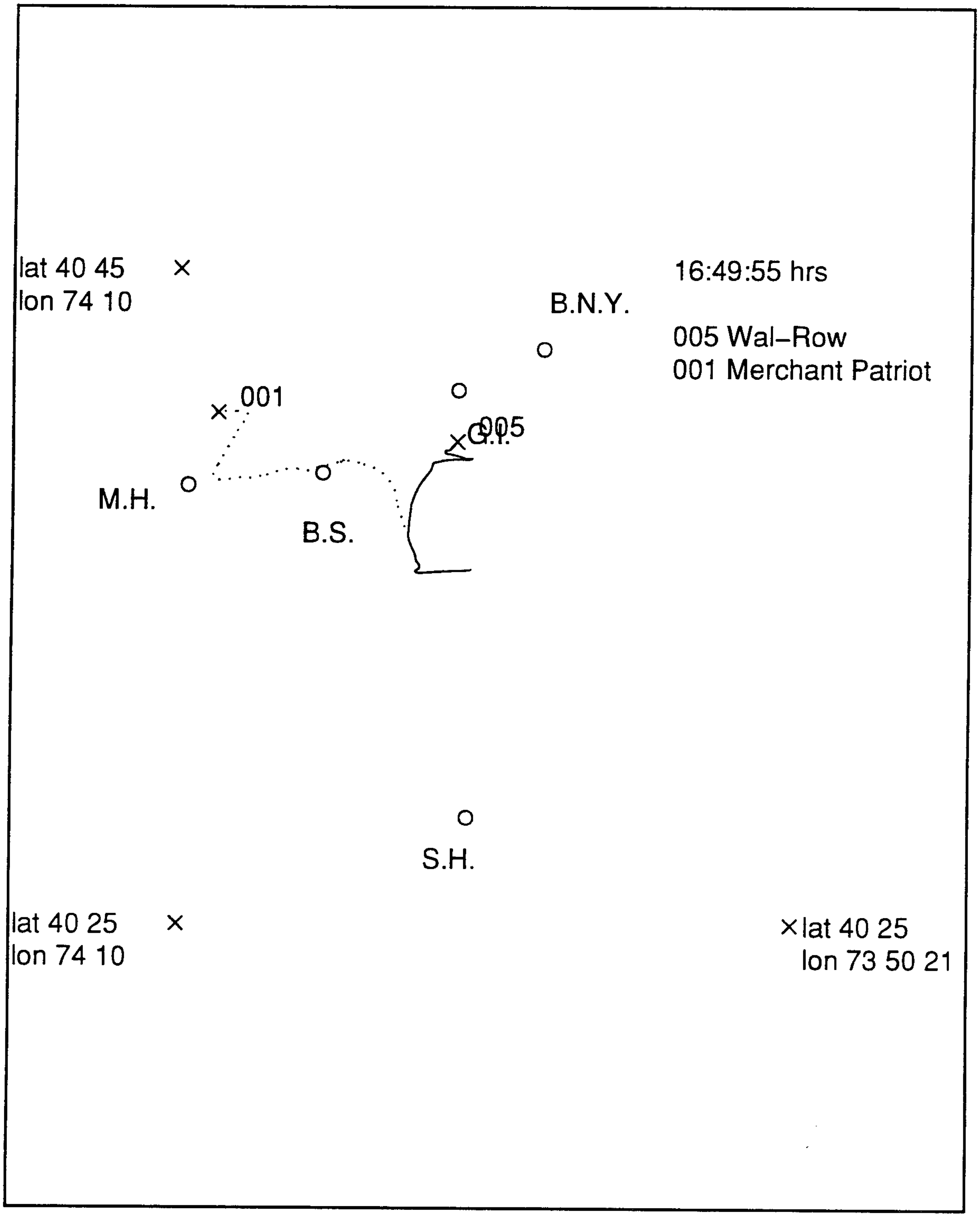

Figure 8. A Typical Plot of Track Fusion for Two Vessels.

V. EVALUATION OF THE FUSION ALGORITHM

This chapter presents the results of various tests performed on the proposed fusion model. The optimum system parameter settings found from extensive simulation are presented first. This is followed by the optimum system parameters to perform four types of fusion tests. Finally, a study on the effects of varying the system parameters on the fusion performance is conducted.

A. SYSTEM PARAMETERS

The performance of the proposed fusion algorithm is essentially controlled by three parameters: the threshold setting, the learning rate parameter, and the radar buffer update rate.

1. Threshold Setting

The best threshold setting found is θ = 100 metres. The threshold setting has a physical interpretation. It is related to the distance between two vessels: one vessel is represented by the input vector, and the other is represented by the weights of the winning neuron. The threshold setting determines if the two vessel tracks belong to the same vessel or two different vessels.

2. Learning Rate Parameter

For the learning rate parameter, $\eta(n)$, as defined in Equation (12), the following values are found to produce the best fusion results: $\eta_i = 0.9$, $\eta_f = 0.1$, and $L = 10$. This means that the learning rate decreases linearly from 0.9 to 0.1 over 10 iterations.

3. Radar Buffer Update Rate

This parameter refers to the average rate at which a radar sends its track data to the VTS system for fusion. The value is assumed to be every 5 seconds and can be

changed as necessary. It is set as a parameter here because studies are conducted here to observe its effect on fusion performance when the rate is increased or decreased.

B. FUSION TESTS

Four types of fusion tests were performed. The first is to test the fusion of each vessel track separately to ensure that the fusion algorithm is functioning properly and also to cllect some statistics of fusion. The second is to test the fusion of all the vessel tracks. The third fusion test is to test the ability of the double fusion resolver to distinguish two vessels that are close together. The last is to test the fusion of single vessel track obtained from different radars with overlaping regions of coverage. Test results are not reproduced here as they are bulky. Instead, the fusion performance is summarized by the fusion accuracy. Fusion accuracy is defined as the number of vessel tracks that are fused correctly over the total number of vessel tracks. Note that a veesel track is either fused correctly or not. Partially correct fusion is the same as incorrect fusion.

1. Fusion of Single Vessel Track

This is the simplest case of fusion test where each simulation involves only one vessel track file. All the track files were tested using the optimum system parameters. Fusion accuracy of 100% is achieved for each vessel track. The plot of track fusion for veesel track file ship14 is shown in Figure 9. Statistical data were collected for each vessel track, and the results are shown in Table 1. For each vessel track file, the maximum fusion distance encountered during fusion is recorded. Fusion distance is the computed distance (using the distance measure defined in Chapter III) between the input data and the weights of the winning neuron, and it must also fall below the threshold setting. The mean and the standard deviation of the fusion distances enountered during the fusion process are also listed.

Vessel Track Files	Maximum Fusion Distance (m)	Mean Fusion Distance (m)	Standard Deviation of Fusion Distance (m)
ship01	33.0	11.9	10.1
ship02	60.8	9.6	12.6
ship03	28.5	6.5	5.8
ship04	69.9	14.5	20.3
ship05	43.2	7.4	9.4
ship06	40.2	10.2	10.0
ship07	19.6	6.3	4.4
ship08	69.3	11.1	14.2
ship09	3.9	2.3	1.2
ship10	27.5	7.2	5.1
ship11	3.9	2.3	1.2
ship12	95.9	12.8	22.8
ship13	23.3	7.9	5.1
ship14	42.3	11.2	10.0
ship15	17.1	4.9	3.9
ship16	20.7	8.2	4.9
ship17	93.7	14.0	21.8
ship18	21.8	5.3	5.1
ship19	44.9	4.5	6.9
ship20	65.5	10.3	12.5
ship21	93.7	11.6	20.0
ship22	3.9	2.4	1.2
ship23	51.6	7.5	8.3
ship24	49.6	6.3	7.7

Table 1. Fusion Statistics for Vessel Track Files.

2. Fusion of All Vessel Tracks

This test includes all the veesel track files in one simulation. Fusion accuracy of 100% is achieved. The maximum fusion distance is 95.9 m, the mean fusion distance is 7.8 m, and the standard deviation is 10.7 m. The fused tracks for ship01 through ship05 are shown in Figure 10.

3. Double Fusion Test

By default, the double fusion resolver is used in the fusion system under test. To demonstrate the ability of the double fusion resolver to distinguish vessel tracks that are close together, the previous test was repeated without the double fusion resolver. Fusion accuracy of 83% is obtained. Incorrect fusion occurs for ship01, ship06, ship07 and ship19. The tracks of ship01 and ship06 are so close together at some point in time that the track of ship06 is fused with the track of ship01 and vice versa. The same problem occurs to ship07 and ship19. This type of fusion problem is more difficult to solve than the problem of crossing tracks. This is because the vessels travel in the same direction and at about the same speed. They are typically less than 10 metres apart. Crossing tracks are relatively easy to distinguish because their directions of travel are different. Figure 11 shows the vessel tracks of ship01 and ship06 without using the double fusion resolver. The same fused tracks using the double fusion resolver are shown in Figure 12. In Figure 11, the track with I.D. 001 is above the track with I.D. 002. This is incorrect. Track 002 should be above track 001 as shown in Figure 12.

4. Fusion of Vessel Tracks from Different Radars

This is the best test of multisensor data fusion. The test is to determine if the tracks of the same vessel obtained from different radars at about the same time are fused. There are three such vessels in the vessel track files: "Steven F. O''Hara" (ship04 and ship22), "Wal-Row" (ship05 and ship11), and "Buchanan 10" (ship17 and ship21). Unfortunately, for "Steven F. O''Hara" and "Wal-Row", radar reports do not

synchronize in time. For example, the last entry of ship05 is reported at 15:38:00 hours, but the first entry of ship11 is reported at 16:07:00 hours (approximately half an hour apart). This means that fusion of the two tracks cannot take place as there is only one observation per vessel at a given time. Ship17 and ship21 are the only two files that have observations on the same vessel at about the same time. Fusion accuracy of 100% is achieved for this pair. The fused track is shown in Figure 13.

C. EFFECT OF VARYING SYSTEM PARAMETERS

1. Effect of Varying the Threshold Setting

To see the effect of using different threshold settings the values of θ = 200 metres and θ = 50 metres are tested (compared to the optimum threshold setting of 100 metres). At the threshold setting of 200 metres, fusion accuracy of 100% is achieved as expected. This is because this value represents a less stringent fusion requirement than the optimum value of 100 metres. But there is a danger in using a less stringent threshold setting. A newly detected vessel that is closer to an earlier detected vessel may be incorrectly fused to the earlier detected vessel, so the objective here is to find a threshold setting that is as small as possible.

Using the threshold setting of 50 metres, however, leads to splitting of tracks and a fusion accuracy of 67%. Examination of the maximum fusion distances in Table 1 reveals that ship02, ship04, ship08, ship12, ship17, ship20, ship21, and ship23 are the tracks that are split due to the fusion distances being greater than the threshold setting. The new VTS system is likely to require a smaller threshold setting than 100 metres because the tracks used in the simulation here are artificially interpolated and smoothed. The smoothed course and speed do not accurately follow the physical laws of motion. In fact, by using the threshold setting of 30 metres for 10 of the 24 vessel tracks with maximum fusion distance of 30 metres or less, a fusion accuracy of 100% is achieved. This compares quite favourably with the GPS data accuracy of 10 metres. More measured data using the upgraded radars need to be collected and tested to determine the best threshold setting for implementation in the new VTS system.

2. Effect of Varying the Learning Rate Parameter

The learning rate parameter is controlled by three variables: the initial learning rate, the final learning rate, and the number of iterations in the training phase of the neural network. As there are many possible combinations of these values, tests are conducted using three different criteria: learning with emphasis on the latter data, learning with emphasis on the earlier data, and fast learning (with short training iteration).

The first test was conducted with $n_i = 0.9$, $n_f = 0.5$, and $L = 10$, and a fusion accuracy of 100% is achieved. The second test was conducted with $n_i = 0.5$, $n_f = 0.1$, and $L = 10$. Fusion accuracy of 100% is again achieved. The last test uses $n_i = 0.9$, $n_f = 0.1$, and $L = 1$. In this case, a fusion accuracy of only 54% is achieved. The results show that the number of iteration, during the training phase has significant effect on the fusion performance. To meet real-time requirements in the VTS system, the neural network is expected to be implemented in hardware to take advantage of its parallel processing capability. To make the training time as effective as possible, training can be performed during the idle time when the system is waiting for the next data buffer to arrive.

3. Effect of Varying the Radar Update Interval

The current radar sensor data update duration is 5 seconds, and this parameter is determined by the specific radar system being used. Tests were performed to observe the effect of varying this parameter. When the update interval is set at 3 seconds, fusion accuracy of 100% is achieved as over-sampling typically helps improve the performance. With a shorter update interval, the threshold setting can also be lowered. When the update interval is set at 10 seconds or more, massive splitting of tracks occurred. For the radar update interval of 10 seconds, a total of 217 tracks are observed instead of 23 vessel tracks; for the case of 15 seconds, 311 tracks are observed. The poor result is caused by the poor prediction of current vessel location based on the previous vessel location. This results in a larger computed

distance which exceeds the threshold setting causing the creation of new tracks. Thus, a shorter radar update interval improves the fusion performance.

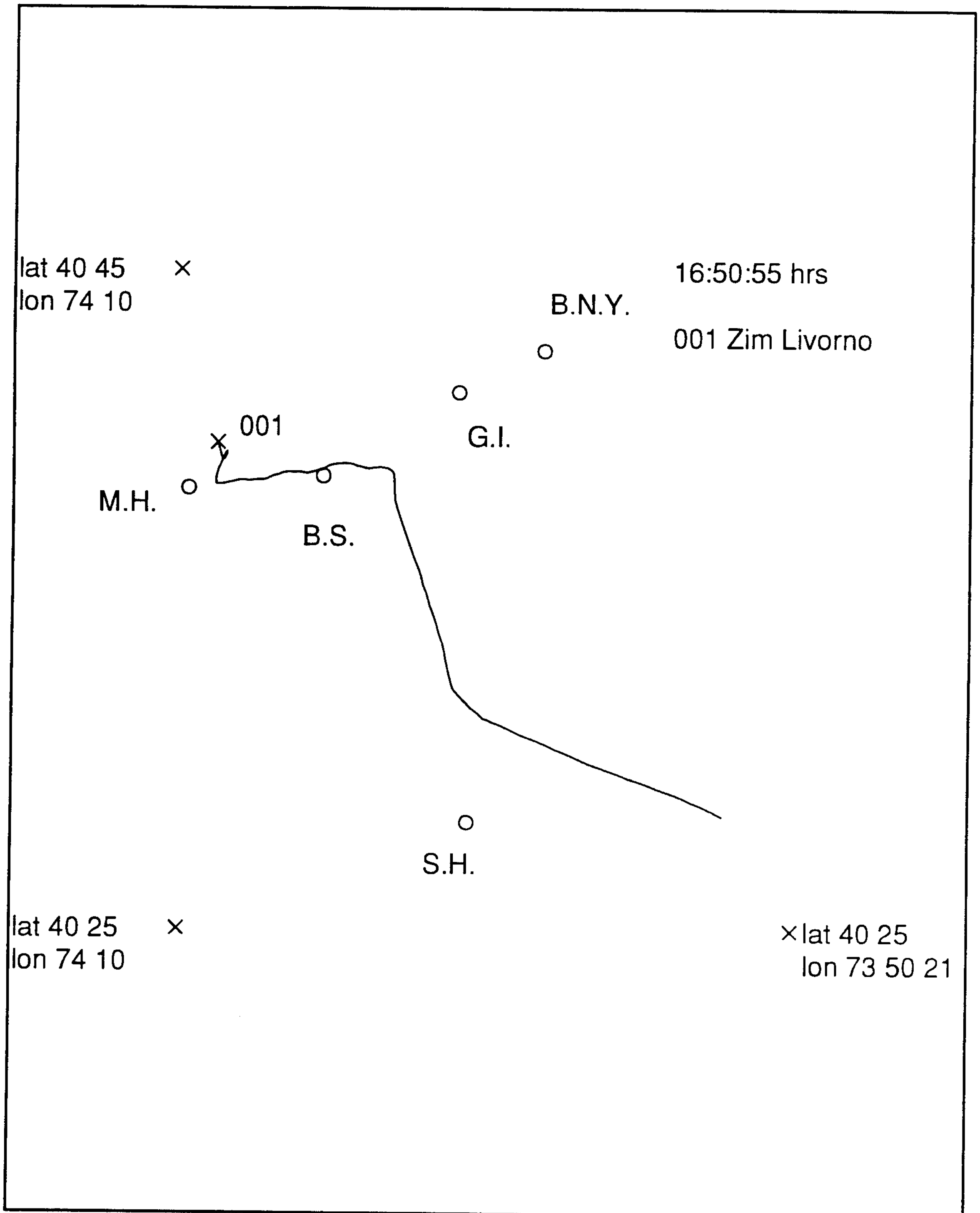

Figure 9. Fusion of Single Vessel Track for Ship14.

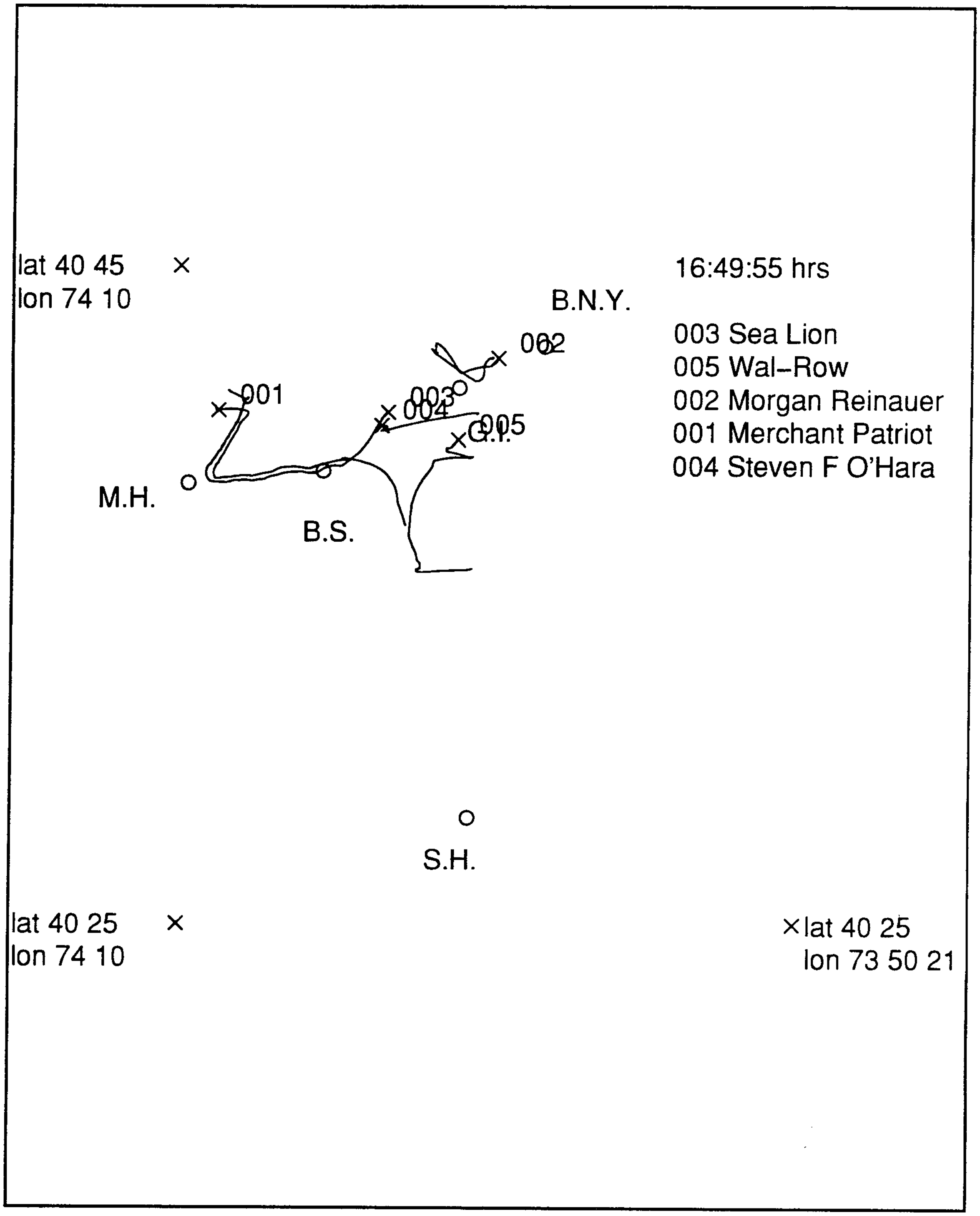

Figure 10. Fusion of Multiple Vessel Tracks.

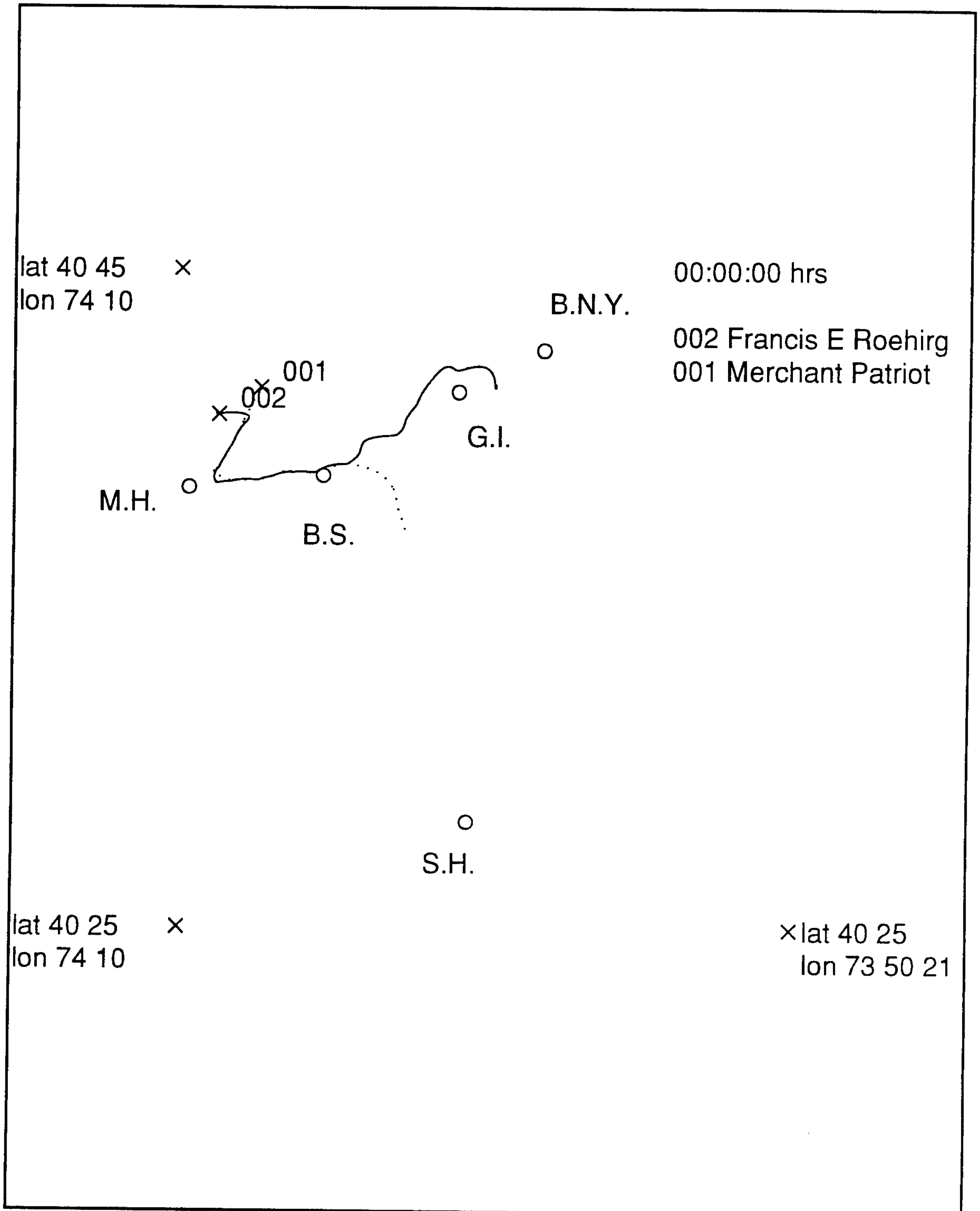

Figure 11. Fusion of Ship01 and Ship06 without Double Fusion Resolution.

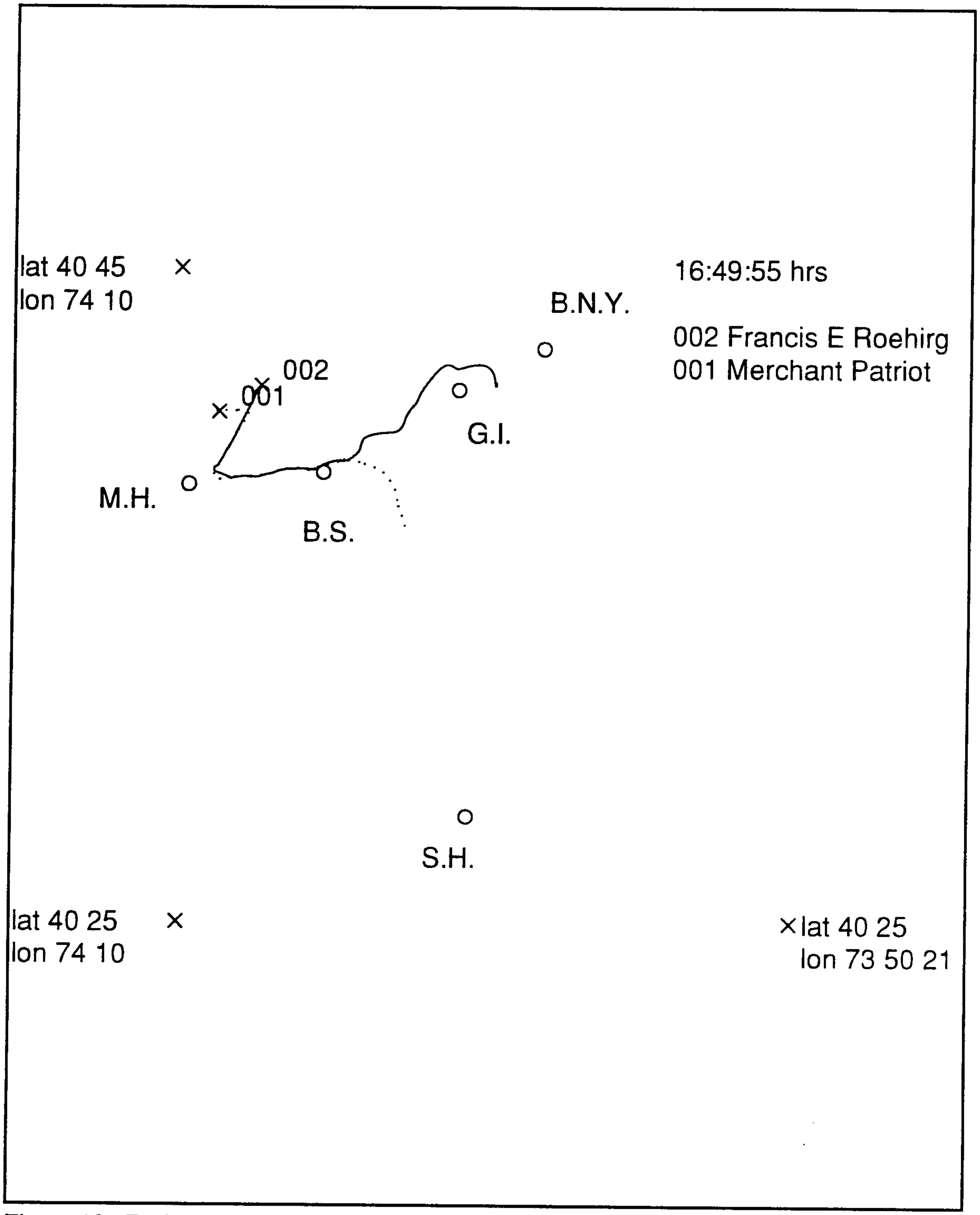

Figure 12. Fusion of Ship01 and Ship06 with Double Fusion Resolution.

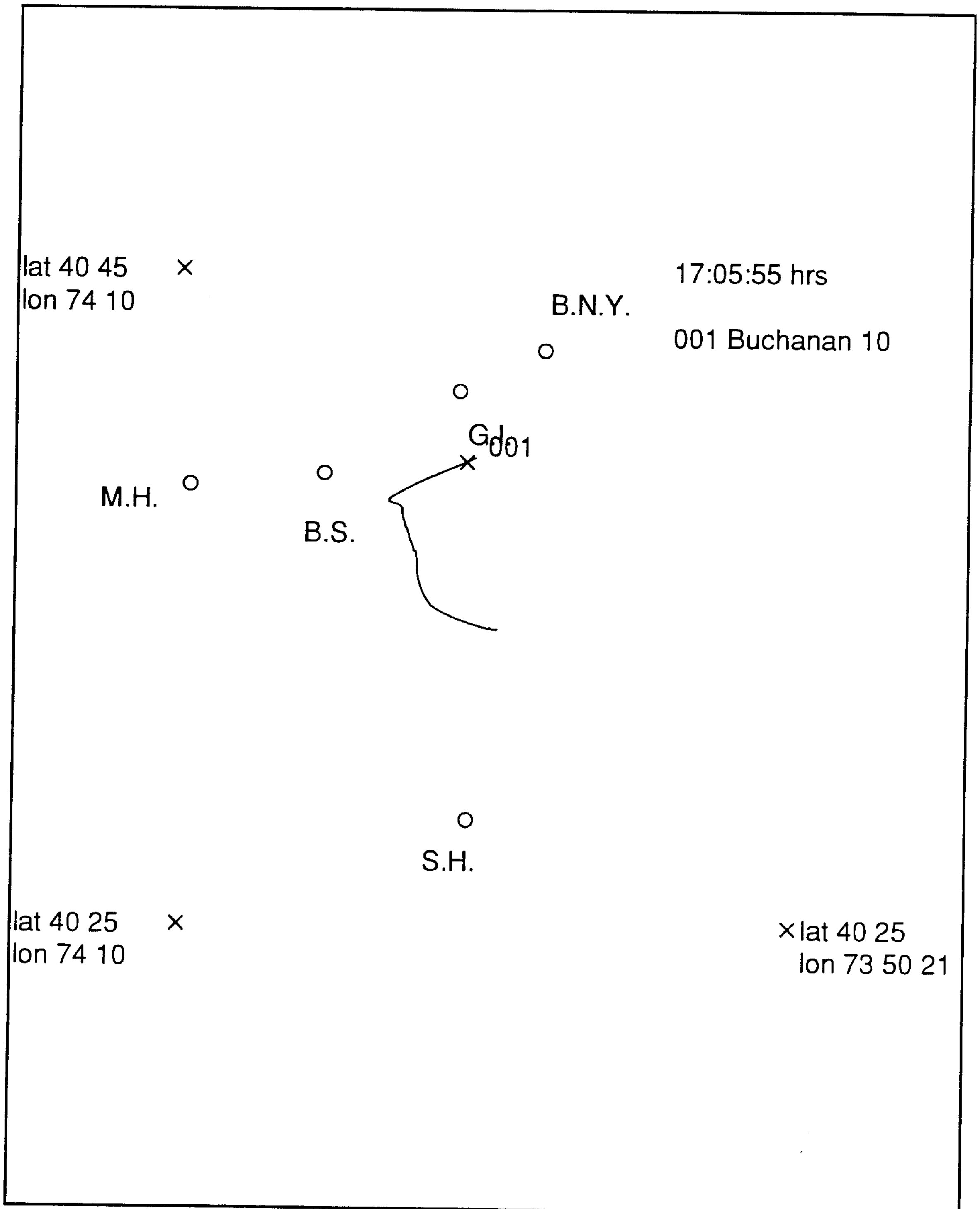

Figure 13. Fusion of Ship17 and Ship19.

VI. CONCLUSION

A. SUMMARY

The goal of this thesis is to explore the use of neural networks in multisensor data fusion for Vessel Traffic Services (VTS). First input data to the VTS system that are suitable for fusion are identified. This was followed by an evaluation of the various neural networks for suitability in the data fusion application. The Kohonen's self-organizing feature map (SOFM) was identified as the most suitable neural network for use in data fusion, but it has some drawbraks that limit its application to solve the VTS data fusion problem. As a result, a new neural network data fusion model was proposed that consists of a modified SOFM and a double fusion resolver to solve the problem of double fusion in VTS. The proposed model was simulated in software and tested with measured input data supplied by the U.S. Coast Guard. The optimum parameters of the fusion system were obtained by experimentation. Using these parameters, fusion accuracies of 100% were consistently achieved for all the tests performed; thus, the proposed neural network fusion model has considerable potential for implementation in the VTS system.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

The proposed fusion model has worked well with the given data. However, more measured data are headed to determine the best system parameters for implementation in the VTS system. In particular, data for slow moving vessels, fast moving vessels, closely spaced vessels, and the same vessel that were sent from different radars and the GPS/DGPS data are needed to adequately validate the fusion performance.

This thesis has covered the level 1 processing functions as described in Chapter II. Research in Level 2 processing may thus be continued. In Level 2 processing, we pursue a higher level of inference which usually means situation assessment. For VTS, this means that the system can automatically detect a potentially dangerous

traffic condition based on the fused data and initiate alarms and corrective instructions to the affected vessels. It seems that a heuristic based approach will be useful at this level of processing.

The proposed system fuses sensor data from multiple radars and GPS. There are other types of sensor data such as video images and audio data, which may provide extra information and this can be considered for fusion. Controlled generation of video data (same aspect angles, no background vessels or other objects etc.) and audio data (extraction of verbally-reported location, speed, and course) may be performed in the laboratory, and these data may be used to experiment with different fusion algorithms.

The proposed model uses the Kohonen's self-organizing feature map. There are other neural networks which may also be suitable for the data fusion application. A relatively new neural network that seems to have some potential is the Grossberg's Adaptive Resonance Theory (ART) neural network [17] and its variants. It will be interesting to study those neural networks and see if they perform as well or better than the SOFM in data fusion applications.

The proposed model cannot be implemented in software in an actual VTS system as the processing speed will be too slow. To gain full benefits of using neural networks, whose power lies in its massive parallelism, a hardware implementation should be considered. Unfortunately, neural network hardware is still in the early stage of development, but this also means that there are many opportunities available for further research on this front.

APPENDIX A. REAL VESSEL TRACK FILES

File Name : ship01
Vessel Name : Merchant Patriot

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201642	404038.4	740846.2	310.2	0.0
201638	404038.4	740846.2	310.2	0.0
201634	404038.4	740846.2	310.2	0.0
201630	404038.4	740846.2	310.2	0.0
201626	404038.4	740846.2	310.2	0.0
201622	404038.4	740846.2	310.2	0.0
201618	404038.4	740846.2	310.2	0.0
201614	404038.4	740846.2	310.2	0.0
201610	404038.4	740846.2	310.2	0.0
201607	404038.4	740846.2	310.2	0.0
201603	404038.4	740846.2	310.2	0.0
201600	404038.4	740846.2	310.2	0.0
201556	404038.4	740846.2	310.2	0.0
201552	404038.4	740846.2	310.2	0.0
201549	404038.4	740846.2	310.2	0.0
201545	404036.2	740752.7	018.9	0.0
201541	404018.2	740800.8	018.9	0.0
201537	404001.1	740811.1	026.1	0.0
201533	403919.8	740836.6	025.0	0.0
201529	403903.2	740846.8	025.0	0.0
201526	403849.5	740855.2	025.0	0.0
201522	403834.6	740851.1	269.0	0.0
201519	403837.4	740822.3	256.6	0.0
201515	403839.3	740741.6	276.4	0.0
201511	403846.8	740703.2	245.5	0.0
201508	403855.5	740633.7	275.2	0.0
201504	403852.8	740553.8	275.2	0.0
201501	403900.5	740525.2	238.6	0.0
201457	403909.1	740439.7	273.0	0.0
201453	403909.6	740450.2	273.0	0.0
201449	403903.2	740404.6	286.1	0.0
201446	403832.9	740316.8	345.5	0.0
201442	403751.9	740302.5	344.6	0.0
201438	403710.5	740247.5	344.6	0.0

File Name : ship02
Vessel Name : Morgan Reinauer

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201535	404212.4	735951.2	38.5	0.0
201531	404212.4	735951.2	38.5	0.0
201528	404210.1	735953.6	38.5	4.0
201524	404158.4	740005.3	25.5	4.0
201521	404149.6	740051.5	90.3	4.0
201517	404149.7	740107.5	90.3	4.0
201513	404238.5	740146.5	190.1	4.0
201509	404220.1	740150.8	190.1	0.0
201505	404220.1	740150.8	190.1	0.0
201502	404220.1	740150.8	190.1	0.0
201458	404220.1	740150.8	190.1	0.0
201454	404220.1	740150.8	190.1	0.0
201450	404220.1	740150.8	190.1	0.0
201447	404220.1	740150.8	190.1	0.0
201443	404130.7	740035.0	202.2	8.0
201439	404158.3	740019.1	221.5	8.0
201436	404212.0	740003.2	221.5	0.0

201435	404212.0	740003.2	221.5	0.0
201432	404212.0	740003.2	221.5	0.0
201429	404212.0	740003.2	221.5	0.0
201428	404212.0	740003.2	221.5	0.0

File Name : ship03
Vessel Name : Sea Lion

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201537	404034.5	740321.4	79.3	4.0
201534	404027.2	740329.5	27.2	4.0
201530	404014.0	740338.4	26.3	4.0
201527	404003.3	740345.3	26.3	4.0
201523	404005.4	740344.0	26.3	7.0
201519	403942.5	740358.9	26.3	7.0
201515	403921.5	740417.7	46.5	7.0
201511	403857.8	740450.2	93.9	7.0
201508	403857.0	740511.8	82.8	7.0
201504	403845.7	740537.3	75.6	7.0
201501	403844.9	740558.0	94.7	7.0
201457	403846.2	740619.0	94.7	8.0
201453	403840.4	740654.7	65.2	6.0
201449	403833.5	740718.9	77.6	6.0
201445	403830.5	740742.2	107.0	6.0
201441	403831.3	740806.3	80.2	6.0
201437	403828.3	740829.2	80.2	6.0
201433	403830.7	740903.1	119.6	4.0
201429	403849.7	740909.6	209.1	4.0
201425	403901.3	740901.1	209.1	4.0
201421	403913.6	740852.1	209.1	4.0
201417	403927.0	740842.3	209.1	4.0
201414	403936.5	740836.0	205.5	4.0
201410	403950.4	740827.2	205.5	4.0
201406	404003.7	740818.8	205.5	4.0
201402	404017.6	740810.1	205.5	4.0
201355	404043.3	740759.6	196.2	4.0
201351	404020.0	740808.6	205.5	0.0
201347	404020.0	740808.6	205.5	0.0
201343	404056.2	740754.3	204.2	0.0
201340	404056.2	740754.3	204.2	0.0
201338	404110.5	740823.4	33.9	0.0
201336	404110.5	740823.4	33.9	0.0

File Name : ship04
Vessel Name : Steven F. O'Hara

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201536	404010.4	740334.4	308.9	4.0
201533	404004.0	740324.0	308.9	4.0
201529	404002.8	740322.1	308.9	4.0
201526	404026.6	740115.8	350.1	0.0
201522	404031.6	740031.9	132.0	0.0
201519	404031.6	740031.9	132.0	0.0
201515	404031.6	740031.9	132.0	0.0
201511	404031.6	740031.9	132.0	0.0
201508	404031.6	740031.9	132.0	0.0

File Name : ship05
Vessel Name : Wal-Row

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201538	403944.7	740107.0	37.8	3.0

201535	403938.5	740113.4	37.8	3.0
201531	403930.0	740122.1	37.8	3.0
201528	403923.8	740128.4	37.8	3.0
201524	403912.8	740046.0	226.7	0.0
201521	403912.8	740046.0	226.7	0.0
201517	403912.8	740046.0	226.7	0.0
201513	403912.8	740046.0	226.7	0.0
201509	403912.8	740046.0	226.7	0.0
201505	403912.8	740046.0	226.7	0.0
201502	403912.8	740046.0	226.7	0.0
201458	403912.8	740046.0	226.7	0.0
201454	403912.8	740046.0	226.7	0.0
201450	403908.6	740143.9	37.8	6.0
201447	403856.1	740156.8	37.8	6.0
201443	403832.3	740215.3	23.8	6.0
201439	403811.6	740227.4	23.8	6.0
201435	403749.7	740235.6	07.7	6.0
201432	403731.3	740237.8	01.9	6.0
201428	403652.5	740241.0	344.6	6.0
201424	403630.0	740232.8	344.6	6.0
201420	403607.6	740225.0	345.3	6.0
201417	403553.0	740220.0	345.3	6.0
201416	403545.5	740217.4	345.3	6.0
201413	403549.5	740050.8	284.2	0.0
201409	403549.5	740050.8	284.2	0.0
201405	403549.5	740050.8	284.2	0.0
201401	403549.5	740050.8	284.2	0.0

File Name : ship06
Vessel Name : Francis E. Roehirg

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201541	404126.6	740725.2	27.5	6.0
201537	404106.8	740738.8	27.5	6.0
201534	404050.7	740746.1	18.9	6.0
201530	403947.9	740819.4	25.0	5.0
201527	403934.3	740827.7	25.0	5.0
201523	403917.3	740838.2	25.0	5.0
201519	403900.7	740848.4	25.0	5.0
201515	403849.9	740855.0	25.0	6.0
201511	403837.3	740823.0	256.6	8.0
201508	403839.2	740812.2	256.6	10.0
201504	403839.6	740732.4	259.3	10.0
201501	403846.7	740703.4	245.5	10.0
201457	403854.7	740622.2	275.2	9.0
201453	403854.0	740543.2	255.8	8.0
201449	403906.5	740512.2	238.6	8.0
201445	403909.2	740441.4	273.0	8.0
201441	403928.6	740416.9	236.5	8.0
201437	403953.8	740400.7	200.9	8.0
201433	404001.8	740306.1	202.3	8.0
201429	404028.3	740248.3	209.8	8.0
201425	404054.2	740228.7	209.8	8.0
201421	404120.7	740214.1	200.1	8.0
201417	404202.1	740134.9	008.0	8.0
201414	404156.3	740108.0	270.3	8.0
201410	404202.4	740014.4	221.5	6.0
201406	404127.0	735956.4	221.5	6.0
201405	404129.6	735954.6	221.5	0.0
201402	404129.6	735954.6	221.5	0.0
201359	404129.6	735954.6	221.5	0.0
201358	404129.6	735954.6	221.5	0.0

File Name : ship07
Vessel Name : Cissi Reinauer

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201541	404010.9	740340.4	26.3	7.0
201537	403947.4	740355.7	26.3	7.0
201533	403925.1	740412.7	46.5	7.0
201529	403858.0	740501.8	82.8	12.0
201526	403845.8	740536.6	75.6	10.0
201522	403846.0	740615.9	94.7	10.0
201518	403840.3	740655.0	65.2	10.0
201514	403830.9	740743.9	107.0	10.0
201510	403829.3	740821.6	80.2	10.0
201507	403826.0	740852.3	119.6	10.0
201503	403844.9	740936.2	119.6	8.0
201459	403844.3	741021.2	87.0	6.0
201455	403843.3	741044.6	87.0	6.0
201452	403841.6	741103.4	72.0	6.0
201451	403818.9	741137.0	34.6	0.0
201448	403818.9	741137.0	34.6	0.0
201444	403818.9	741137.0	34.6	0.0
201440	403818.9	741137.0	34.6	0.0
201436	403818.9	741137.0	34.6	0.0
201432	403818.9	741137.0	34.6	0.0

File Name : ship08
Vessel Name : Fright

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201602	403939.1	740112.8	37.8	4.0
201559	403930.6	740121.5	37.8	4.0
201555	403919.7	740132.6	37.8	4.0
201551	403908.6	740143.9	37.8	4.0
201548	404021.7	740226.4	27.1	8.0
201544	403954.5	740244.7	27.1	8.0
201540	403929.0	740301.4	27.1	8.0
201536	403844.4	740400.4	114.8	10.0
201532	403857.0	740436.4	114.8	10.0
201528	403852.2	740522.5	57.1	10.0
201524	403844.7	740555.9	94.7	8.0
201521	403842.9	740647.5	65.2	0.0
201517	403842.9	740647.5	65.2	0.0
201513	403845.1	740638.0	79.7	0.0
201509	403845.1	740638.0	79.7	0.0
201508	403845.1	740638.0	79.7	0.0
201506	403907.8	740640.5	278.5	0.0
201502	403907.8	740640.5	278.5	0.0
201459	403907.8	740640.5	278.5	0.0

File Name : ship09
Vessel Name : Catherine Brown

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201607	404055.5	740836.7	33.9	0.0
201603	404055.5	740836.7	33.9	0.0
201600	404055.5	740836.7	33.9	0.0
201556	404055.5	740836.7	33.9	0.0
201552	404055.5	740836.7	33.9	0.0
201549	404055.5	740836.7	33.9	0.0
201545	404055.5	740836.7	33.9	0.0
201541	404055.5	740836.7	33.9	0.0
201537	404055.5	740836.7	33.9	0.0
201534	404055.5	740836.7	33.9	0.0
201530	404055.5	740836.7	33.9	0.0
201527	404055.5	740836.7	33.9	0.0
201523	404055.5	740836.7	33.9	0.0

201519	404055.5	740836.7	33.9	0.0
201515	404055.5	740836.7	33.9	0.0
201511	404055.5	740836.7	33.9	0.0
201508	404055.5	740836.7	33.9	0.0
201504	404055.5	740836.7	33.9	0.0
201501	404055.5	740836.7	33.9	0.0
201457	404055.5	740836.7	33.9	0.0
201453	404055.5	740836.7	33.9	0.0
201449	404055.5	740836.7	33.9	0.0
201445	404055.5	740836.7	33.9	0.0
201441	404055.5	740836.7	33.9	0.0
201437	404055.5	740836.7	33.9	0.0
201433	404055.5	740836.7	33.9	0.0
201429	404055.5	740836.7	33.9	0.0
201425	404055.5	740836.7	33.9	0.0
201421	404055.5	740836.7	33.9	0.0
201417	404055.5	740836.7	33.9	0.0
201414	404055.5	740836.7	33.9	0.0
201410	404055.5	740836.7	33.9	0.0
201406	404055.5	740836.7	33.9	0.0
201402	404055.5	740836.7	33.9	0.0
201358	404055.5	740836.7	33.9	0.0
201355	404055.5	740836.7	33.9	0.0
201351	404055.5	740836.7	33.9	0.0
201347	404055.5	740836.7	33.9	0.0
201343	404055.5	740836.7	33.9	0.0
201340	404055.5	740836.7	33.9	0.0
201336	404055.5	740836.7	33.9	0.0
201332	404055.5	740836.7	33.9	0.0
201328	404055.5	740836.7	33.9	0.0
201324	404055.5	740836.7	33.9	0.0
201321	404055.5	740836.7	33.9	0.0
201317	404055.5	740836.7	33.9	0.0
201313	404055.5	740836.7	33.9	0.0
201309	404055.5	740836.7	33.9	0.0
201305	404055.5	740836.7	33.9	0.0
201301	404055.5	740836.7	33.9	0.0
201257	404055.5	740836.7	33.9	0.0
201253	404055.5	740836.7	33.9	0.0
201249	404055.5	740836.7	33.9	0.0
201245	404055.5	740836.7	33.9	0.0
201242	404055.5	740836.7	33.9	0.0
201238	404055.5	740836.7	33.9	0.0
201234	404055.5	740836.7	33.9	0.0
201231	404055.5	740836.7	33.9	0.0
201230	404055.5	740836.7	33.9	0.0
201229	404055.5	740836.7	33.9	0.0

File Name : ship10
Vessel Name : Franklin Reinauer

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201622	403848.9	741058.5	262.7	7.0
201618	403851.0	741031.0	268.6	7.0
201614	403851.5	741002.1	268.6	7.0
201610	403851.6	740935.1	292.3	7.0
201607	403844.8	740914.9	302.3	7.0
201603	403834.6	740849.2	269.0	7.0
201600	403840.4	740758.9	268.4	7.0
201556	403839.9	740730.1	259.3	7.0
201552	403846.7	740703.5	245.5	7.0
201549	403853.6	740643.6	245.5	7.0
201545	403854.3	740615.7	275.2	7.0
201541	403853.1	740548.2	255.8	7.0
201537	403902.1	740521.8	238.6	7.0
201533	403909.4	740447.3	273.0	7.0

201529	403908.9	740432.5	273.0	7.0
201526	403904.5	740410.5	286.1	7.0
201522	403925.4	740325.8	202.3	7.0
201519	403943.2	740316.1	202.3	7.0
201515	404008.5	740302.5	202.3	7.0
201511	404042.7	740237.4	209.8	0.0
201508	404042.7	740237.4	209.8	0.0
201504	404042.7	740237.4	209.8	0.0
201501	404042.7	740237.4	209.8	0.0
201457	404054.0	740133.2	265.1	8.0
201453	404103.9	740105.0	236.9	8.0
201449	404107.9	740056.9	236.9	10.0
201445	404137.2	740031.5	202.2	10.0
201441	404208.6	740007.1	221.5	10.0
201440	404229.3	740005.0	241.4	0.0
201437	404229.3	740005.0	241.4	0.0
201433	404229.3	740005.0	241.4	0.0
201431	404229.3	740005.0	241.4	0.0
201430	404229.3	740005.0	241.4	0.0

File Name : ship11
Vessel Name : Wal-Row

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201628	403926.6	740026.6	226.7	0.0
201625	403926.6	740026.6	226.7	0.0
201621	403926.6	740026.6	226.7	0.0
201618	403926.6	740026.6	226.7	0.0
201614	403926.6	740026.6	226.7	0.0
201610	403926.6	740026.6	226.7	0.0
201607	403926.6	740026.6	226.7	0.0

File Name : ship12
Vessel Name : Nan McKay

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201628	403901.0	740102.4	226.7	0.0
201625	403901.0	740102.4	226.7	0.0
201621	403901.0	740102.4	226.7	0.0
201617	404035.1	740316.8	79.3	5.0
201613	404024.9	740331.0	27.2	5.0
201609	404036.1	740346.9	134.6	0.0
201609	404036.1	740346.9	134.6	0.0
201606	404036.1	740346.9	134.6	0.0
201602	404036.1	740346.9	134.6	0.0
201559	404036.1	740346.9	134.6	0.0
201555	404036.1	740346.9	134.6	0.0
201551	404036.1	740346.9	134.6	0.0
201548	404036.1	740346.9	134.6	0.0

File Name : ship13
Vessel Name : Stephen Reinauer

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201630	403849.3	741054.6	262.7	8.0
201626	403851.2	741022.0	268.6	8.0
201623	403851.6	740958.0	268.6	8.0
201620	403851.3	740934.1	292.3	8.0
201616	403839.6	740904.0	302.3	8.0
201612	403839.5	740903.8	302.3	8.0
201609	403834.8	740842.0	269.0	8.0
201605	403839.7	740809.4	256.6	8.0

201601	403841.5	740719.4	259.3	8.0
201557	403852.0	740648.3	245.5	8.0
201553	403854.4	740617.4	275.2	8.0
201550	403852.8	740553.5	275.2	8.0
201546	403857.0	740532.8	238.6	8.0
201542	403909.9	740457.7	273.0	7.0
201538	403908.8	740430.1	273.0	7.0
201535	403904.3	740409.6	286.1	7.0
201531	403913.0	740330.7	183.4	7.0
201528	403931.6	740322.4	202.3	7.0
201524	404007.0	740303.2	202.3	7.0
201521	404023.7	740251.7	209.8	7.0
201517	404053.1	740148.5	265.1	7.0
201513	404053.6	740140.5	265.1	7.0
201509	404059.4	740114.0	236.9	7.0
201506	404108.4	740055.8	236.9	7.0
201502	404128.7	740036.1	202.2	7.0
201459	404142.0	740015.7	25.5	0.0
201458	404142.0	740015.7	25.5	0.0
201455	404142.0	740015.7	25.5	0.0
201452	404142.0	740015.7	25.5	0.0
201451	404142.0	740015.7	25.5	0.0

File Name : ship14
Vessel Name : Zim Livorno

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201635	403949.0	740848.4	26.4	0.0
201631	403917.9	740837.8	25.0	7.0
201627	403930.2	740830.3	25.0	7.0
201624	403911.1	740841.9	25.0	8.0
201621	403834.6	740850.5	269.0	8.0
201617	403838.1	740818.4	256.6	8.0
201613	403839.8	740748.1	276.4	8.0
201609	403842.2	740716.5	245.5	8.0
201606	403850.5	740652.4	245.5	8.0
201602	403854.7	740621.3	275.2	8.0
201559	403853.0	740557.0	275.2	8.0
201555	403859.7	740527.0	238.6	8.0
201551	403909.9	740457.7	273.0	8.0
201548	403906.9	740421.2	286.1	6.0
201544	403901.8	740358.2	286.1	6.0
201540	403857.6	740313.7	13.6	1.0
201536	403803.2	740306.6	344.6	13.0
201533	403619.0	740229.0	345.3	13.0
201529	403531.1	740212.4	345.3	13.0
201526	403451.8	740158.8	345.3	13.0
201522	403403.3	740142.0	345.3	13.0
201518	403313.2	740128.0	348.0	13.0
201514	403221.2	740113.5	348.0	13.0
201510	403145.9	740039.4	322.3	13.0
201508	403126.6	740016.3	296.8	14.6
201505	403120.0	735959.0	298.2	14.9
201502	403057.4	735904.8	300.6	14.7
201458	403031.0	735757.6	297.1	15.0
201454	403004.5	735652.6	292.8	15.1
201450	402936.6	735532.5	295.1	14.2
201450	402936.6	735532.5	295.1	14.2
201447	402918.0	735438.3	298.6	13.6
201443	402854.2	735335.5	300.8	11.6
201439	402827.6	735234.9	310.1	0.0

File Name : ship15
Vessel Name : Terror

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201635	403953.7	740845.4	26.4	0.0
201631	403953.7	740845.4	26.4	0.0
201627	403953.7	740845.4	26.4	0.0
201624	403953.7	740845.4	26.4	0.0
201621	403953.7	740845.4	26.4	0.0
201617	403953.7	740845.4	26.4	0.0
201613	403953.7	740845.4	26.4	0.0
201609	403942.1	740822.9	25.0	0.0
201606	403942.1	740822.9	25.0	0.0
201602	403942.1	740822.9	25.0	0.0
201559	403937.4	740825.8	25.0	4.0
201555	403924.1	740834.0	25.0	4.0
201551	403910.6	740842.2	25.0	4.0
201548	403900.3	740848.6	25.0	4.0
201544	403837.2	740859.0	302.3	6.0
201540	403834.8	740836.5	269.0	6.0
201536	403839.1	740812.6	256.6	6.0
201533	403840.4	740754.4	276.4	6.0
201529	403839.8	740731.1	259.3	6.0
201526	403843.6	740712.4	245.5	6.0
201522	403851.4	740650.0	245.5	6.0
201519	403855.5	740633.3	275.2	6.0
201516	403854.8	740623.6	275.2	6.0
201515	403906.2	740626.3	278.5	0.0
201511	403906.2	740626.3	278.5	0.0
201508	403906.2	740626.3	278.5	0.0

File Name : ship16
Vessel Name : Newtown Creek

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201638	404204.6	735959.3	38.5	0.0
201634	404149.6	740041.6	90.3	8.0
201630	404149.7	740113.0	90.3	8.0
201626	404228.2	740148.9	190.1	8.0
201624	404259.1	740141.7	190.1	8.0
201622	404300.1	740141.4	190.1	0.0
201619	404300.1	740141.4	190.1	0.0
201615	404300.1	740141.4	190.1	0.0

File Name : ship17
Vessel Name : Buchanan 10

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201639	403909.6	740050.4	226.7	0.0
201636	403909.6	740050.4	226.7	0.0
201632	403909.6	740050.4	226.7	0.0
201628	403909.6	740050.4	226.7	0.0
201625	403909.6	740050.4	226.7	0.0
201621	403909.6	740050.4	226.7	0.0
201618	403909.6	740050.4	226.7	0.0
201614	403909.6	740050.4	226.7	0.0
201610	403809.1	740308.7	345.5	6.0
201607	403751.9	740302.6	344.6	6.0
201603	403729.4	740254.4	344.6	6.0
201600	403712.4	740248.2	344.6	6.0
201556	403649.1	740239.8	344.6	6.0
201552	403626.8	740231.7	345.3	6.0
201549	403609.8	740225.8	345.3	6.0
201545	403446.5	740156.9	345.3	6.0
201541	403404.4	740002.3	265.1	0.0
201537	403404.4	740002.3	265.1	0.0

201534	403404.4	740002.3	265.1	0.0
201530	403404.4	740002.3	265.1	0.0

File Name : ship18
Vessel Name : Eastern Sun

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201639	403850.1	741045.9	262.7	11.0
201636	403851.4	741012.4	268.6	11.0
201632	403856.0	740905.0	209.1	11.0
201628	403931.9	740838.9	205.5	11.0
201625	403957.4	740822.8	205.5	11.0
201621	404035.6	740802.6	196.2	11.0
201617	404115.1	740742.5	208.5	11.0
201613	404150.3	740717.3	212.9	0.0
201609	404150.3	740717.3	212.9	0.0
201605	404150.3	740717.3	212.9	0.0
201601	404150.3	740717.3	212.9	0.0
201557	404150.3	740717.3	212.9	0.0
201553	404150.3	740717.3	212.9	0.0
201550	404150.3	740717.3	212.9	0.0
201546	404150.3	740717.3	212.9	0.0
201542	404150.3	740717.3	212.9	0.0
201538	404150.3	740717.3	212.9	0.0

File Name : ship19
Vessel Name : Chem Trader

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201642	403903.5	740702.5	232.7	0.0
201638	403903.5	740702.5	232.7	0.0
201635	403903.5	740702.5	232.7	0.0
201631	403903.5	740702.5	232.7	0.0
201627	403903.5	740702.5	232.7	0.0
201624	403903.5	740702.5	232.7	0.0
201621	403903.5	740702.5	232.7	0.0
201617	403903.5	740702.5	232.7	0.0
201613	403903.5	740702.5	232.7	0.0
201609	403903.5	740702.5	232.7	0.0
201606	403903.5	740702.5	232.7	0.0
201602	403903.5	740702.5	232.7	0.0
201559	403903.5	740702.5	232.7	0.0
201555	403903.5	740702.5	232.7	0.0
201551	403903.5	740702.5	232.7	0.0
201548	403903.5	740702.5	232.7	0.0
201544	403903.5	740702.5	232.7	0.0
201540	403903.5	740702.5	232.7	0.0
201536	403903.5	740702.5	232.7	0.0
201532	403903.5	740702.5	232.7	0.0
201528	403903.5	740702.5	232.7	0.0
201524	403903.5	740702.5	232.7	0.0
201521	403846.3	740621.6	94.7	11.0
201517	403834.1	740715.7	77.6	9.0
201513	403831.0	740808.7	80.2	9.0
201509	403826.4	740843.9	80.2	9.0
201506	403824.6	740930.7	89.2	0.0
201503	403824.6	740930.7	89.2	0.0
201502	403824.6	740930.7	89.2	0.0
201458	403824.6	740930.7	89.2	0.0

File Name : ship20
Vessel Name : Dean Reinauer

DTG	Latitude	Longitude	Course	Speed

(dd/hh/mm)	(deg./min./sec.)	(deg./min./sec.)	(deg.)	(knots)
201647	404156.6	740006.5	25.5	0.0
201643	404156.6	740006.5	25.5	0.0
201639	404138.9	740017.6	25.5	8.0
201636	404188.4	740030.9	45.2	8.0
201632	404059.6	740055.9	45.2	8.0
201628	404048.1	740124.8	81.5	8.0
201625	404041.6	740212.9	27.1	8.0
201621	404015.4	740230.6	27.1	8.0
201617	403948.6	740248.7	27.1	8.0
201613	403923.1	740305.6	13.6	8.0
201609	403852.8	740424.3	114.8	8.0
201606	403857.8	740449.0	93.9	8.0
201602	403845.5	740538.3	75.6	8.0
201559	403845.1	740602.5	94.7	8.0
201555	403845.7	740633.6	79.7	8.0
201551	403832.2	740726.9	77.6	8.0
201548	403826.9	740840.3	80.2	8.0
201544	403834.2	740911.2	119.6	8.0
201540	403845.3	740939.7	90.0	8.0
201536	403844.7	741012.1	86.4	8.0
201532	403842.4	741100.3	72.0	8.0
201528	403828.0	741128.0	50.2	8.0
201526	403816.6	741139.1	34.6	0.0
201524	403816.6	741139.1	34.6	0.0
201521	403816.6	741139.1	34.6	0.0
201517	403816.6	741139.1	34.6	0.0
201513	403816.6	741139.1	34.6	0.0
201512	403816.6	741139.1	34.6	0.0

File Name : ship21
Vessel Name : Buchanan 10

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201658	403909.6	740050.4	226.7	0.0
201655	403909.6	740050.4	226.7	0.0
201651	403909.6	740050.4	226.7	0.0
201647	403909.6	740050.4	226.7	0.0
201643	403909.6	740050.4	226.7	0.0
201639	403909.6	740050.4	226.7	0.0
201636	403909.6	740050.4	226.7	0.0
201632	403909.6	740050.4	226.7	0.0
201628	403909.6	740050.4	226.7	0.0
201625	403909.6	740050.4	226.7	0.0
201621	403909.6	740050.4	226.7	0.0
201618	403909.6	740050.4	226.7	0.0
201614	403909.6	740050.4	226.7	0.0
201610	403809.1	740308.7	345.5	6.0
201607	403751.9	740302.6	344.6	6.0
201603	403729.4	740254.4	344.6	6.0
201600	403712.4	740248.2	344.6	6.0
201556	403649.1	740239.8	344.6	6.0
201552	403626.8	740231.7	345.3	6.0
201549	403609.8	740225.8	345.3	6.0
201545	403446.5	740156.9	345.3	6.0
201541	403404.4	740002.3	265.1	0.0
201537	403404.4	740002.3	265.1	0.0
201534	403404.4	740002.3	265.1	0.0
201530	403404.4	740002.3	265.1	0.0

File Name : ship22
Vessel Name : Steven F. O'Hara

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)

201702	404034.7	740344.9	134.6	0.0
201658	404034.7	740344.9	134.6	0.0
201655	404034.7	740344.9	134.6	0.0
201651	404034.7	740344.9	134.6	0.0
201647	404034.7	740344.9	134.6	0.0
201643	404034.7	740344.9	134.6	0.0
201639	404034.7	740344.9	134.6	0.0

File Name : ship23
Vessel Name : Itco XII

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201708	404150.2	740906.7	295.0	0.0
201704	404150.2	740906.7	295.0	0.0
201700	404104.9	740739.7	18.9	6.0
201657	404048.7	740747.0	18.9	6.0
201653	404026.6	740757.0	18.9	6.0
201649	404005.0	740808.5	26.1	6.0
201645	403945.7	740820.7	25.0	6.0
201641	403943.3	740822.2	25.0	6.0
201638	403927.0	740832.2	25.0	6.0
201634	403906.8	740844.6	25.0	6.0
201630	403835.7	740855.8	302.3	6.0
201626	403835.7	740831.6	256.6	6.0
201622	403839.9	740808.2	256.6	6.0
201618	403839.5	740744.6	276.4	6.0
201614	403841.4	740720.0	259.3	6.0
201610	403848.6	740658.0	245.5	6.0
201607	403847.4	740701.4	245.5	6.0
201603	403855.3	740638.7	245.5	6.0
201600	403854.6	740620.9	275.2	6.0
201556	403852.9	740556.1	275.2	6.0
201552	403856.9	740533.1	238.6	6.0
201549	403904.4	740516.7	238.6	6.0
201545	403909.7	740453.7	273.0	6.0
201541	403919.0	740435.9	236.5	6.0
201540	403916.6	740440.8	236.5	6.0
201537	403923.5	740427.1	236.5	6.0
201533	403937.8	740408.8	200.9	6.0
201529	403940.7	740317.5	202.3	6.0
201525	404010.2	740301.5	202.3	6.0
201522	404021.1	740253.7	209.8	8.0
201518	404022.7	740252.5	209.8	8.0
201514	404049.1	740232.5	209.8	8.0
201510	404115.0	740216.9	200.1	8.0
201507	404137.4	740206.1	200.1	8.0
201503	404207.6	740153.8	190.1	8.0
201459	404238.7	740146.4	190.1	8.0
201455	404259.3	740141.6	190.1	0.0
201451	404259.3	740141.6	190.1	0.0
201448	404259.3	740141.6	190.1	0.0
201445	404259.3	740141.6	190.1	0.0
201444	404259.3	740141.6	190.1	0.0
201440	404259.3	740141.6	190.1	0.0
201439	404259.3	740141.6	190.1	0.0
201436	404259.3	740141.6	190.1	0.0
201432	404259.3	740141.6	190.1	0.0
201429	404259.3	740141.6	190.1	0.0

File Name : hip24
Vessel Name : ritannia Mcallister

DTG (dd/hh/mm)	Latitude (deg./min./sec.)	Longitude (deg./min./sec.)	Course (deg.)	Speed (knots)
201731	403821.9	741143.2	208.9	0.0

201728	403821.9	741143.2	208.9	0.0
201724	403821.9	741143.2	208.9	0.0
201720	403851.4	741012.1	268.6	4.0
201717	403851.6	741000.3	268.6	4.0
201713	403851.9	740944.1	268.6	4.0
201709	403849.3	740928.0	292.3	4.0
201705	403844.0	740913.2	302.3	4.0
201701	403837.3	740859.2	302.3	4.0
201657	403834.7	740843.6	269.0	4.0
201654	403835.8	740831.4	256.6	4.0
201650	403838.6	740815.4	256.6	4.0
201646	403840.4	740800.1	268.4	4.0
201642	403840.4	740800.2	268.4	4.0
201638	403839.5	740743.7	276.4	4.0
201634	403840.2	740728.0	259.3	4.0
201630	403843.5	740712.8	245.5	4.0
201626	403848.8	740657.3	245.5	4.0
201622	403854.0	740642.3	245.5	4.0
201618	403855.1	740626.9	275.2	4.0
201614	403853.9	740610.3	275.2	4.0
201610	403852.9	740554.9	275.2	4.0
201607	403854.1	740542.9	255.8	4.0
201603	403859.3	740527.9	238.6	4.0
201600	403904.3	740517.0	238.6	4.0
201556	403910.0	740501.7	273.0	4.0
201552	403905.9	740416.9	286.1	4.0
201549	403903.4	740405.2	286.1	4.0
201545	403908.5	740428.4	286.1	4.0
201541	403905.1	740413.0	286.1	4.0
201537	403859.6	740336.0	276.3	4.0
201533	403858.3	740319.9	276.3	4.0
201529	403857.0	740304.4	276.3	4.0
201526	403855.9	740251.6	276.3	4.0
201522	403854.6	740235.8	276.3	4.0
201519	403908.1	740201.4	219.8	0.0
201515	403908.1	740201.4	219.8	0.0
201511	403908.1	740201.4	219.8	0.0
201508	403908.1	740201.4	219.8	0.0
201504	403908.1	740201.4	219.8	0.0
201501	403908.1	740201.4	219.8	0.0
201457	403908.1	740201.4	219.8	0.0
201453	403908.1	740201.4	219.8	0.0
201449	403858.2	740106.3	226.7	0.0
201445	403858.2	740106.3	226.7	0.0
201441	403858.2	740106.3	226.7	0.0
201437	403858.2	740106.3	226.7	0.0
201433	403858.2	740106.3	226.7	0.0

APPENDIX B. MATLAB CODES

Part 1 : Sensor Data Simulation Programs

The program files are : vtss.m, convert1.m, and cnysmo.m.

```
% File Name : vtss.m
% Thesis : Multi-sensor data fusion through neural network
% Student : Koh, Leonard Phin-Liong
% Purpose : main program to generate sensor data and perform simulation.

clear all

% Simulation Variables
N_neuron = 0 ;
max_time = 10 ;

% for testing purposes
dist_cnt = 0 ;

% Literal declaration
TRUE = 1 ;

% plot New York harbour
init_trk ;
koh1ny ;

% select time resolution : min. or sec.
sec_select = 1 ; %seconds resolution
global sec_select ;
if (sec_select ~= 1) , sec_select = 0 ; end

% load program parameters
fusepara ;
dataform ;

% select time resolution : min. or sec.
if sec_select == 1
  showplot = 0 ; % no plots; set to 1 if require plotting
elseif sec_select == 0
  showplot = 1 ;
end

% LOAD INPUT DATA FILE

% load user assigned ships for simulation
sim_name = 'simall' ;
eval (['load ', sim_name]) ;
eval (['shiplist = ', sim_name]) ;
```

```
 eval (['clear ', sim_name]) ;
 global shiplist ;
 num_ship = length (shiplist) ;

 for  nship = 1:num_ship
   if shiplist (nship) < 10
     shipfile = ['ship0' num2str(shiplist(nship))] ;
   else
     shipfile = ['ship' num2str(shiplist(nship))] ;
   end

   eval (['load ', shipfile]) ;
   eval (['SHIP = ', shipfile, ';']) ;

   SHIP = flipud (SHIP) ;

   SHIP = convert (SHIP, sec_select) ;
   filesize = size (SHIP) ;
   shipfilesize (nship, 1) = filesize (1) ;
   filestarttime (nship, 1) = SHIP (1, raw_time_i) ;
   filestoptime  (nship, 1) = SHIP (filesize(1), raw_time_i) ;
   eval ([shipfile, ' = SHIP ;']) ;
 end

 starttime = min (filestarttime) ;
 stoptime  = max (filestoptime) ;

% clear some workspace
clear SHIP convert cny dtg2time num2grid inc_sec

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% fptr is  a list of individual buffer ptr for each shipfile
% note that each shipfile has different length
fptr = ones (num_ship, 1) ;

if sec_select == 0
  time_inc = 100 ;
elseif sec_select == 1
  time_inc = 5 ;
end

for ft = starttime:time_inc:stoptime
  % initialise radar buffer
  bufi = 1 ;
  buf = zeros (BUF_SIZE, NUM_BUF_FIELD) ;

  for nship = 1:num_ship
    if shiplist(nship) < 10
      shipfile = ['ship0' num2str(shiplist(nship))] ;
```

```
  else
    shipfile = ['ship' num2str(shiplist(nship))] ;
  end

  % check fptr () > 0 before loading in the time
  if fptr (nship) > 0
    eval (['SHIP = ', shipfile, '(', num2str(fptr(nship)) ,', :) ;']) ;

    ftime = SHIP (1, raw_time_i) ;
    if ftime == ft
      buf (bufi, buf_lat_i)    = SHIP (1, raw_lat_i) ;
      buf (bufi, buf_long_i)   = SHIP (1, raw_long_i) ;
      buf (bufi, buf_course_i) = SHIP (1, raw_course_i) ;
      buf (bufi, buf_speed_i)  = SHIP (1, raw_speed_i) ;
      buf (bufi, buf_time_i)   = SHIP (1, raw_time_i) ;
      buf (bufi, buf_shipid_i) = nship ;

      if fptr (nship, 1) == shipfilesize (nship, 1) ;
        fptr (nship) = 0 ;
      else
        fptr (nship) = fptr (nship) + 1 ;
      end

      bufi = bufi + 1 ;
    end % if ftime
  end % valid fptr (nship)
end % for each shipfile

% send one buffer for fusion
if bufi > 1
% buffer is not empty
  % delete unused buffer space
  buf = buf (1:bufi-1, :) ;

  % simulate GPS info.
  prob = rand ;
  if prob > 0.7
     buf (:, buf_sensor_i) = GPS_SENSOR .* ones (bufi-1, 1) ;
  end

  % convert and format input data buffer

  if N_neuron > 0
    fusion ;
  else
    X = buf (1, :) ;
    add_new ;
    % if there is multiple radar starting at same time, consider add FUSION
    % here to fuse the rest of first buffer.
  end
```

```
    [hh, mm, ss] = time2sec (ft) ;
    if (showplot == 1) & (ss == 0)
      plottrk ;
    end
  end
% clear buf
end

% Collect statistics for simulation
disp (['max. fusion distance is      ' num2str(max(best_dist))]) ;
disp (['Averaged fusion distance is ' num2str(mean(best_dist))]) ;
disp (['sigma of fusion distance is  ' num2str(std(best_dist))]) ;
disp (['Median of fusion distance is ' num2str(mean(best_dist))]) ;
disp (' ')
W
W(:, id_i)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function s = convert (ship, second_resolution) ;
% File Name : convert1.m
% Purpose : convert and format input data buffer : SHIP

dataform ;

ship_size = size (ship) ;
ship_size = ship_size (1) ;

if second_resolution == 0
% no need to interpolate data : time resolution is minutes.
  s = zeros (ship_size, NUM_RAW_FIELD) ;

  for j = 1:ship_size
    s (j, raw_time_i) = dtg2time (ship (j, raw_time_i)) ;
    s (j, raw_lat_i) = num2grid (ship (j, raw_lat_i)) ;
    s (j, raw_long_i) = num2grid (ship (j, raw_long_i)) ;
    s (j, raw_course_i) = ship (j, raw_course_i) ;
    s (j, raw_speed_i) = ship (j, raw_speed_i) ;
  end

elseif second_resolution == 1
% interpolate data : time resolution is seconds.
   s = cnysmo (ship) ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function tracks=createny(ship) ;
% File Name : cnysmo.m
% function creates the dbts of tracks based on given start and stop positions and speed
% the generated track will consist of [x, y, crs, spd, time]', the tracks will be
% in 5 sec increments
```

```
inc = 5;      %increment=5 sec
avg_min = 4 ;  % assumed average no. minutes between entry.
incsize = 60*avg_min/inc ;
L = 3 ;       % length of original data for interpolation >= 2*L

%for tracks
dataform ;
INVALID_COURSE = 999 ;

speed = interp (ship(:,raw_speed_i), incsize, L) ;

load map_ref.dat ;
ref_long = num2grid (map_ref(1)) ;
ref_lat  = num2grid (map_ref(2)) ;

% compute distance between the 2 given points in nm
shiplen  = size (ship) ;
shiplen  = shiplen (1) ;
rlong = ones (shiplen, 1) * ref_long ;
rlat  = ones (shiplen, 1) * ref_lat  ;
long_entry = num2grid (ship(:, raw_long_i)) ;
lat_entry  = num2grid (ship(:, raw_lat_i)) ;
[xx, yy] = lonlat2k (rlong, rlat, long_entry, lat_entry) ;

y = interp (yy, incsize, L) ;
x = interp (xx, incsize, L) ;

y = y (1:(shiplen-1)*incsize) ;
x = x (1:(shiplen-1)*incsize) ;
rlong = ones ((shiplen-1)*incsize, 1) * ref_long ;
rlat  = ones ((shiplen-1)*incsize, 1) * ref_lat  ;
[long lat] = km2lonla (rlong, rlat, x, y) ;

tracks = zeros ((shiplen-1)*incsize, 5) ;

tracks (:, raw_long_i) = long ;
tracks (:, raw_lat_i)  = lat ;
tracks (:, raw_speed_i)  = speed (1:(shiplen-1)*incsize) ;

tracks (1, raw_time_i) = dtg2time (ship(1,raw_time_i)) ;
for i = 2:(shiplen-1)*incsize
  tracks (i, raw_time_i) = inc_sec (tracks(i-1, raw_time_i), inc) ;
end

for i = 2:shiplen
  crs = ship(i-1, raw_course_i) ;
  tracks ((i-2)*incsize+1, raw_course_i) = ship(i-1, raw_course_i) ;
  crs_d = crs_diff (crs, ship(i,raw_course_i)) / incsize ;

  for n = 2:incsize
```

```
    crs = crs + crs_d ;
    if crs >= 360
      crs = crs - 360 ;
    elseif crs < 0
      crs = 360 + crs ;
    end
    tracks ((i-2)*incsize+n, raw_course_i) = crs ;
  end
end
```

Part 2 : NEURAL NETWORK SIMULATION PROGRAMS

The program files are : dataform.m, fusepara.m, fusion.m, fusedist.m, finefuse.m, add_new.m, train.m.

```
% File Name : dataform.m
% Data Format Definition

% General Literals
UNUSED = 0 ;
USED   = 1 ;
UNUSED_DISTANCE = 99999 ;

global UNUSED_DISTANCE

% raw data input
% variable name : SHIP
% 1    2     3      4       5
% time lat.  long.  course.  speed
% no. of buffer data fields
NUM_RAW_FIELD = 5 ;
% time : 0 - 2359 hrs

raw_time_i   = 1 ;
raw_lat_i    = 2 ;
raw_long_i   = 3 ;
raw_course_i = 4 ;
raw_speed_i  = 5 ;

% buffer input
% variable name : buf
% 1     2    3       4      5    6      7
% long. lat. course. speed time sensor ship_id
% no. of buffer data fields
NUM_BUF_FIELD = 7 ;
% time : 0 - 2359 hrs /2400 : invalid
INVALID_TIME = 2400 ;
% sensor_type : 0 - radar  1 - GPS
RADAR_SENSOR = 0 ;
```

```
GPS_SENSOR   = 1 ;
% ship_id : a ASCII no.from GPS info. / 0 : invalid
INVALID_ID   = 0 ;

buf_long_i    = 1 ;
buf_lat_i     = 2 ;
buf_course_i  = 3 ;
buf_speed_i   = 4 ;
buf_time_i    = 5 ;
buf_sensor_i  = 6 ;
buf_shipid_i  = 7 ;

% format for neuron weights
% variable name : W
% 1    2     3       4      5    6  7        8   9
% long. lat. course. speed valid ID valid_ID TFI TLI
% course: 0-360 deg wrt North
% speed: >= 0
% ID : a ASCII no.from GPS info. / 0 : invalid
% valid_id : 1 : ID is valid (update from GPS)  0 : not valid
% TFI : 0000 to 2359 hrs
% TLI : 0000 to 2359 hrs / 2400 : not valid

long_i     = 1 ;
lat_i      = 2 ;
course_i   = 3 ;
speed_i    = 4 ;
used_i     = 5 ;
id_i       = 6 ;
valid_id_i = 7 ;
tfi_i      = 8 ;
tli_i      = 9 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File Name : fusepara.m
% Thesis simulation variable

BUF_SIZE = 30 ; % each radar can handle up to 60 targets

if sec_select == 1
  % for simallc
  DISTANCE_THRESHOLD = 0.1 ;
else
  DISTANCE_THRESHOLD = 3 ;
end

max_time_gap = 600 ; % 10 min

DELTA_SPEED_THRESHOLD = 5 ; % 5 knots
```

```
global DISTANCE_THRESHOLD DELTA_SPEED_THRESHOLD max_time_gap

% finefuse.m para
FINEFUSE_KM_TOL  = 0.02 ; % formerly 0.4 km
FINEFUSE_CRS_TOL = 10 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File Name : fusion.m
% Neural network fusion is mainly performed here.


% INITIALIZE VARIABLES BEFORE FOR-LOOP
% neuron is a vector used for checking double fusion & record buffer index
neuron = zeros (N_neuron, 3) ;

% for counting occurrence of double fusion
double_fusion_count = 1 ;

% set number of new neurons added for this batch of input buffer to 0
num_new_neuron = 0 ;

buffer_size = size (buf) ;
for n = 1:buffer_size(1),
  % get the input pattern
  X = buf (n, :) ;

  fprintf ('time=%-4.0f ', buf(n,buf_time_i)) ;
  fprintf ('id=%-2.0f ', buf(n,buf_shipid_i)) ;

  % determine the wining neuron
  for i = 1:N_neuron
    if W(i, used_i) == USED
      % can change norm to weighted sensor distance weighting
      distance (i,1) = fusedist (W(i, long_i), W(i, lat_i), W(i, course_i), ...
                                 W(i, speed_i), W(i, tli_i), ...
                                 X(1, buf_long_i), X(1, buf_lat_i), ...
                                 X(1, buf_course_i), X(1, buf_speed_i), ...
                                 X(1, buf_time_i) ) ;
    else
    % this neuron is not used
      distance (i,1) = UNUSED_DISTANCE ;
    end
  end
  [n_list, list_idx] = sort (distance) ;
  fprintf ('id=%-2.0f ', W (list_idx(1),id_i)) ;
  fprintf ('d=%-2.4f\n\n', n_list(1)) ;
  if n_list(1) <= DISTANCE_THRESHOLD
    % fusion is successful
    % mark winning neuron to detect double fusion
```

```
dist_cnt = dist_cnt + 1 ;
best_dist (dist_cnt,1) = n_list(1) ;
%disp (' ');

   if neuron (list_idx(1), 1) == 1,
   % double fusion occurs. note neuron and activate 2nd network later
%test
disp (['double fusion occurs at time = ' num2str(buf(n, buf_time_i)) ]) ;

     dub_neuron = list_idx(1) ;
     neuron (dub_neuron, 1) = neuron (dub_neuron, 1) + 1 ;
     neuron (dub_neuron, 3) = n ;

     finefuse ;

     X = buf (rej_buf_n, :) ;
     for dd = 1:N_neuron
       if (W(dd, used_i) == USED) & (dd ~= dub_neuron)
         distance (dd,1) = fusedist (W(dd,long_i),W(dd,lat_i), ...
                                     W(dd,course_i), ...
                                     W(dd, speed_i), W(dd, tli_i), ...
                                     X(1, buf_long_i), X(1, buf_lat_i), ...
                                     X(1, buf_course_i), ...
                                     X(1, buf_speed_i), ...
                                     X(1, buf_time_i) ) ;
       else
       % this neuron is not used or is the double fused neuron
         distance (dd,1) = UNUSED_DISTANCE ;
       end
     end
     [n_list, list_idx] = sort (distance) ;
     if n_list(1) <= DISTANCE_THRESHOLD
       if neuron (list_idx(1), 1) == 1
       % 2nd level double fusion occurs
         dub_neuron = list_idx (1) ;
         neuron (dub_neuron, 1) = neuron (dub_neuron, 1) + 1 ;
         neuron (dub_neuron, 3) = rej_buf_n ;

         finefuse ;

         X = buf (rej_buf_n, :) ;
         for ddd = 1:N_neuron
           if (W(ddd, used_i) == USED) & (neuron (ddd,1) == 0)
           % only look for neuron that has not been fused before
             distance (ddd,1) = fusedist (W(ddd,long_i),W(ddd,lat_i), ...
                                          W(ddd,course_i), ...
                                          W(ddd, speed_i), W(ddd, tli_i), ...
                                          X(1, buf_long_i), X(1, buf_lat_i), ...
                                          X(1, buf_course_i), ...
                                          X(1, buf_speed_i), ...
```

```
                                       X(1, buf_time_i) ) ;
          else
          % this neuron is not used or is the double fused neuron
            distance (ddd,1) = UNUSED_DISTANCE ;
          end
        end
        [n_list, list_idx] = sort (distance) ;
        if n_list(1) <= DISTANCE_THRESHOLD
        % this is the first time this neuron has won; mark as one match and
        % record buffer index
          neuron (list_idx(1), 1) = 1 ;
          neuron (list_idx(1), 2) = rej_buf_n ;
        else
        % double fusion 3rd try fails ==> possible new vessel
%test
disp ('double fusion 3rd try fails : add new neuron') ;
          % add new neuron
          add_new ;
        end
      else
        % this is the first time this neuron has won; mark as one match and
        % record buffer index
        neuron (list_idx(1), 1) = 1 ;
        neuron (list_idx(1), 2) = rej_buf_n ;
      end
    else
    % double fusion 2nd try fails ==> possible new vessel
%test
disp ('double fusion 2nd try fails : add new neuron') ;
disp (' ')
      % add new neuron
      add_new ;
    end
  else
    % this is the first time this neuron has won; mark as one match and
    % record buffer index
    neuron (list_idx(1), 1) = 1 ;
    neuron (list_idx(1), 2) = n ;
  end; % double fusion handling

  % update of weights is done in one step at the last.
 else
 % fusion fails ==> possible new vessel
  % add new neuron
  add_new ;

  % note no. of new neurons added
  num_new_neuron = num_new_neuron + 1 ;
 end; % if for threshold checking
end; % for loop for input buffer
```

```
% now perform batch update of weights
train ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function delta_km = fusedist (w_long, w_lat, w_crs, w_spd, w_time, ...
                              b_long, b_lat, b_crs, b_spd, b_time) ;
% File Name : fusedist.m
% This function computes the distance between a weight vector and an input
% vector for Kohonen's SOFM neural network.

% w_ : weight vector ; b_ : buffer input vector

global DELTA_SPEED_THRESHOLD  sec_select UNUSED_DISTANCE max_time_gap

% compute time difference in seconds
[hr1 min1 sec1] = time2sec (w_time) ;
[hr2 min2 sec2] = time2sec (b_time) ;

if b_time < w_time
% pass 2400 hrs
  t_sec = etime ([0 0 1 hr2 min2 sec2], [0 0 0 hr1 min1 sec1]) ;
else
  t_sec = etime ([0 0 0 hr2 min2 sec2], [0 0 0 hr1 min1 sec1]) ;
end

% compute acceleration/deceleration ?

% estimate distance travelled by vessel over t_sec
distance = nm2km ( (w_spd / 3600) * t_sec ) ;

distance_x = distance * cos ( radians (90-w_crs) ) ;
distance_y = distance * sin ( radians (90-w_crs) ) ;

% predict/estimate current position of W's vessel
[w_est_long w_est_lat] = km2lonla (w_long, w_lat, distance_x, distance_y) ;

% compute distance between estimated W position and buffer position
[km_x km_y] = lonlat2km (w_est_long, w_est_lat, b_long, b_lat) ;
delta_km = sqrt (km_x^2 + km_y^2) ;

% if buffer time is too far away from neuron last update time, don't fuse
if t_sec > max_time_gap
  delta_km = UNUSED_DISTANCE ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File Name : finefuse.m
% perform finer fusing.
% dub_neuron is the neuron with double fusion
% rej_buf_n : output- the ousted buffer entry which lost the fusion.
```

```
% compare position

X1 = buf (neuron (dub_neuron, 2), :) ;
distance1 = fusedist (W(dub_neuron, long_i), W(dub_neuron, lat_i), ...
                      W(dub_neuron, course_i), W(dub_neuron, speed_i), ...
                      W(dub_neuron, tli_i), ...
                      X1(1, buf_long_i), X1(1, buf_lat_i), ...
                      X1(1, buf_course_i), X1(1, buf_speed_i), ...
                      X1(1, buf_time_i) ) ;

X2 = buf (neuron (dub_neuron, 3), :) ;
distance2 = fusedist (W(dub_neuron, long_i), W(dub_neuron, lat_i), ...
                      W(dub_neuron, course_i), W(dub_neuron, speed_i), ...
                      W(dub_neuron, tli_i), ...
                      X2(1, buf_long_i), X2(1, buf_lat_i), ...
                      X2(1, buf_course_i), X2(1, buf_speed_i), ...
                      X2(1, buf_time_i) ) ;

disp(['finefuse distance of buffer 1 = ' num2str(distance1)]) ;
disp(['finefuse distance of buffer 2 = ' num2str(distance2)]) ;

if abs (distance1 - distance2) <= FINEFUSE_KM_TOL
% the 2 distance is too close. finer matching must continue.
  crs_err1 = min (abs (W(dub_neuron, course_i) - X1(1,buf_course_i)), ...
             abs (360 - (W(dub_neuron, course_i) - X1(1,buf_course_i)))) ;

  crs_err2 = min (abs (W(dub_neuron, course_i) - X2(1,buf_course_i)), ...
             abs (360 - (W(dub_neuron, course_i) - X2(1,buf_course_i)))) ;

  if abs (crs_err1 - crs_err2) <= FINEFUSE_CRS_TOL
  % the 2 course is too close. finer matching must continue.

    speed_err1 = abs (W (dub_neuron, speed_i) - X1 (1, buf_speed_i)) ;
    speed_err2 = abs (W (dub_neuron, speed_i) - X2 (1, buf_speed_i)) ;

    if speed_err1 < speed_err2
      finefuse_winner = 1 ;
 disp(['finefuse winner is the earlier buffer entry due to speed']);
    elseif speed_err1 > speed_err2
      finefuse_winner = 2 ;
 disp(['finefuse winner is the later buffer entry due to speed']);
    elseif speed_err1 == speed_err2
      finefuse_winner = 1 ;
      if W (dub_neuron, valid_id_i) == 1
        if X1 (1, buf_shipid_i) == W (dub_neuron, id_i)
          finefuse_winner = 1 ;
 disp(['finefuse winner is the earlier buffer entry due to id']);
        elseif X2 (1, buf_shipid_i) == W (dub_neuron, id_i)
          finefuse_winner = 2 ;
 disp(['finefuse winner is the later buffer entry due to id']);
```

```
        end
      else
  disp(['finefuse winner is the earlier buffer entry due to arbit. choice']);
      end
    end
  else
  % the 2 course is too far apart. One of them has to be rejected.
    if crs_err1 < crs_err2
      finefuse_winner = 1 ;
    else
      finefuse_winner = 2 ;
    end
if finefuse_winner == 1
  disp(['finefuse winner is the earlier buffer entry due to course']);
elseif finefuse_winner == 2
  disp(['finefuse winner is the later buffer entry due to course']);
end
end

else
% the 2 distance is too far apart. One of them has to be rejected.
  if distance1 < distance2
    finefuse_winner = 1 ;
  else
    finefuse_winner = 2 ;
  end
if finefuse_winner == 1
  disp(['finefuse winner is the earlier buffer entry due to distance']);
elseif finefuse_winner == 2
  disp(['finefuse winner is the later buffer entry due to distance']);
end
end

if finefuse_winner == 1
  rej_buf_n = neuron (dub_neuron, 3) ;
else
  rej_buf_n = neuron (dub_neuron, 2) ;
end

% housekeep variable neuron
neuron (dub_neuron, 1) = 1 ;
if neuron (dub_neuron, 2) == rej_buf_n
  neuron (dub_neuron, 2) = neuron (dub_neuron, 3) ;
end

neuron (dub_neuron, 3) = 0 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File Name : add_new.m
% add new neurons
```

```
% Literals

% make sure new neuron's vessels does not appear from nowhere;
% it must be from furthest boundary of the radar
% 2 conditions for new vessel detection :
% a) first arrival at the port
% b) non-overlap radar coverage ==> faulty intermediate radar or non-overlap
%     region too far apart to be fused together ==> must detect & combine them.
%     pass thru' fusing network to find best match. check 'new' vessel and best
%     match correlate in terms of time difference and distance/direction
%     travelled.
%     may be use expectation fusing network to do this

% scan from start to locate unused neurons
unused_neuron = UNUSED ;
for new = 1:N_neuron,
  if W (new, used_i) == UNUSED,
    unused_neuron = new ;
    break ;
  end
end; % for new

if unused_neuron == UNUSED,
% have to add one new neuron
  if N_neuron == 0
    W_size = 0 ;

    % put buf (n) = buf (1)
    n = 1 ;
  else
    W_size = size (W) ;
    W_size = W_size(1) ;
  end

  W (W_size+1, lat_i)    = X (1, buf_lat_i) ;
  W (W_size+1, long_i)   = X (1, buf_long_i) ;
  W (W_size+1, course_i) = X (1, buf_course_i) ;
  W (W_size+1, speed_i)  = X (1, buf_speed_i) ;
  W (W_size+1, used_i)   = USED ;
  W (W_size+1, id_i)     = X (1, buf_shipid_i) ;
  W (W_size+1, valid_id_i) = 0 ;
  if X (1, buf_sensor_i) > RADAR_SENSOR,
  % buf info is from GPS/DGPS
    W (W_size+1, valid_id_i) = 1 ;
  end
  W (W_size+1, tfi_i)    = X (1, buf_time_i) ;
  W (W_size+1, tli_i)    = X (1, buf_time_i) ; % formerly = INVALID_TIME ;

  % create double fusion detection entry for new neuron
  neuron (W_size+1, 1) = 0 ;
```

```
  N_neuron = N_neuron + 1 ;
 % test
 disp ('new neuron added') ;

  % plot fused vessel track
  winner = W_size+1 ;
  add_trk ;
  storetrk ;
 else
 % unused neuron found ==> reuse it as a 'new' neuron
  W (unused_neuron, lat_i)    = X (1, buf_lat_i) ;
  W (unused_neuron, long_i)   = X (1, buf_long_i) ;
  W (unused_neuron, course_i) = X (1, buf_course_i) ;
  W (unused_neuron, speed_i)  = X (1, buf_speed_i) ;
  W (unused_neuron, used_i)   = USED ;
  W (unused_neuron, id_i)     = X (1, buf_shipid_i) ;
  W (unused_neuron, valid_id_i) = 0 ;
  if X (1, buf_sensor_i) > RADAR_SENSOR,
  % buf info is from GPS/DGPS
    W (unused_neuron, valid_id_i) = 1 ;
  end
  W (unused_neuron, tfi_i)    = X (1, buf_time_i) ;
  W (unused_neuron, tli_i)    = X (1, buf_time_i) ; % formerly = INVALID_TIME ;

  % create double fusion detection entry for new neuron
  neuron (unused_neuron, 1) = 0 ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File Name : train.m
% TRAINING PHASE

NONE = 0 ;

for upi = 1:N_neuron

if neuron (upi, 1) == 1
  winner = upi ;
  X = buf (neuron (upi, 2), :) ;
else
  winner = NONE ;
end

if winner ~= NONE
% time is updated by simply overwriting time of last intercept
W (winner, tli_i) = X (1, buf_time_i) ;

if X(1, buf_sensor_i) > RADAR_SENSOR
% info. is from GPS/DGPS ==> just replace the current info
  W (winner, lat_i) = X (1, buf_lat_i) ;
```

```
    W (winner, long_i) = X (1, buf_long_i) ;
    W (winner, course_i) = X (1, buf_course_i) ;
    W (winner, speed_i) = X (1, buf_speed_i) ;
    if W (winner, valid_id_i) == 0
      W (winner, id_i)       = X (1, buf_shipid_i) ;
      W (winner, valid_id_i) = 1 ;
    end
  else
  % info. is from radar
    for k = 1:max_time
    % corrupt pattern randomly with a given probability
    % X is not necessary
    % X = buf (n, :) ;

      % compute the current learning rate
      lrn_rate = max (0.9 - 0.8 * k / max_time, 0.1) ;

      % update the wining neuron and note the difference of old & new weight
      W (winner, 1:4) = W (winner, 1:4) .* (1 - lrn_rate) + ...
                     lrn_rate .* X (1, 1:4)  ;

    % W_diff = norm (W (winner, :) - W_old (winner, :)) ;
    end
  end

  % store fused vessel track
  storetrk ;

  end
  end
```

Part 3 : PLOTTING AND DISPLAY PROGRAMS

The program files are : init_trk.m, add_trk.m, storetrk.m, plottrk.m, koh1ny.m, plottype.m, id2ship.m.

```
% File Name : init_trk.m
% init variable for add_trk.m, storetrk.m and plot_trk.m

if N_neuron > 0
  trk_idx (N_neuron) = ones (N_neuron) ;

  trk_long = zeros (N_neuron, 1) ;
  trk_lat  = zeros (N_neuron, 1) ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File Name : add_trk.m
% add additional track entry
```

```
trk_idx (length(trk_idx)+1) = 1 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File Name : storetrk.m
% store individual fused vessel track
% before this routine is called, winner variable must be initialized.

trk_long (winner, trk_idx (winner)) = W (winner, long_i) ;
trk_lat  (winner, trk_idx (winner)) = W (winner, lat_i) ;

trk_idx (winner) = trk_idx (winner) + 1 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File Name : plottrk.m
% plot vessel tracks

%figure(1)
clf
koh1ny

current_time = buf (1, buf_time_i) ;
[hh, mm, ss] = time2sec (current_time) ;
hr = num2str (hh) ;  mm = num2str (mm) ;  ss = num2str (ss) ;
if length(hr) == 1, hr = ['0' hr] ; end
if length(mm) == 1, mm = ['0' mm] ; end
if length(ss) == 1, ss = ['0' ss] ; end
%text (15, 20, ['          '] ) ;
text (12, 20, [hr ':' mm ':' ss ' hrs'] ) ;

% initialize starting position for ship labelling
label_y = 18 ; % previously 10

load map_ref.dat ; % -741000  402500
ref_long = num2grid (map_ref(1)) ;
ref_lat  = num2grid (map_ref(2)) ;

trk_size = size (trk_long) ;
trk_size = trk_size (1) ;

for trk = 1:trk_size
  trk_length = trk_idx (trk) - 1 ;

  if trk_length > 0
    for i = 1:trk_length
      [mpx mpy] = lonlat2k(ref_long, ref_lat, trk_long(trk,i), trk_lat(trk,i));
      mpp = km2nm([mpx mpy]);

      trk_x (trk, i) = mpp(1) ;
      trk_y (trk, i) = mpp(2) ;
    end
```

```
      plot_symbol = plottype (trk) ;
      plot (trk_x(trk,1:trk_length), trk_y(trk,1:trk_length), plot_symbol)
      % show current point differently
      last_x = trk_x (trk, trk_length) ;
      last_y = trk_y (trk, trk_length) ;
      plot (last_x, last_y, 'xw')
      text (last_x + 0.5, last_y + 0.5, [ship_num(W(trk, id_i))] ) ;
      if W (trk, valid_id_i) > 0
        text (12, label_y, [ship_num(W(trk, id_i)) ' ' id2ship(W(trk, id_i))]) ;
      else
        text (12, label_y, [ship_num(W(trk, id_i)) ' ' id2ship(0)]) ;
      end
      label_y = label_y - 1 ;
    end % trk_length > 0
  end

  if sec_select == 0
    pause (2)
  elseif sec_select == 1
    drawnow ;
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y = srny(x)
% File Name : koh1ny.m
% This function plots the picture of the NY harbour
% x and y are dummy variables

% 1 nm = 1852 m = 1.151 mi

% Radar sites coverage in terms of radius
bny_rad = 2;
bs_rad  = 3;
gi_rad  = 3;
mh_rad  = 3;
sh_rad  = 7;

% Geographical Locations of Radar sites
ref            = [40+25/60 74+10/60];
brooklyn_yard  = [40+42/60+33.9/3600 73+58/60+22/3600];
bank_street    = [40+38/60+48.52/3600 74+5/60+25.33/3600];
gov_island     = [40+41/60+18.6/3600 74+1/60+5.53/3600];
mariners_harbor = [40+38/60+27.08/3600 74+9/60+43.7/3600];
sandy_hook     = [40+28/60+15.37/3600 74+44.89/3600];

[bny_x bny_y]=lonlat2k(74+10/60, 40+25/60, 73+58/60+22/3600, 40+42/60+33.9/3600);
[bs_x bs_y]=lonlat2k(74+10/60, 40+25/60, 74+5/60+25.33/3600, 40+38/60+48.52/3600);
[gi_x gi_y]=lonlat2k(74+10/60, 40+25/60, 74+1/60+5.53/3600,40+41/60+18.6/3600);
[mh_x mh_y]=lonlat2k(74+10/60, 40+25/60, 74+9/60+43.7/3600, 40+38/60+27.08/3600);
[sh_x sh_y]=lonlat2k(74+10/60, 40+25/60, 74+44.89/3600, 40+28/60+15.37/3600);
```

```
bny=km2nm([bny_x bny_y]);
bs=km2nm([bs_x bs_y]);
gi=km2nm([gi_x gi_y]);
mh=km2nm([mh_x mh_y]);
sh=km2nm([sh_x sh_y]);

figure(1)
v=[-5 15 -5 20];
axis(v); axis('square'), axis('off'), hold on, axis(axis)

%%%% RADAR SITES
plot (-sh(1), sh(2), 'wo', -bs(1), bs(2), 'wo', -mh(1), mh(2), 'wo', ...
    -gi(1), gi(2), 'wo', -bny(1), bny(2), 'wo')

text(6,2,'S.H.') ;     % text(4,3,'Sandy')
                       % text(4.2,2, 'Hook')
text(-2,13,'M.H.') ;   % text(-4,13,'Mariners')
                       % text(-4,12, 'Harbor')
text(3,12,'B.S.') ;  % text(1.5,12,'Bank')
                       % text(1.5,11, 'Street')
text(7,15,'G.I.') ;   % text(10,16,'Govenors Island')
text(9,19,'B.N.Y.') ; % text(10,18,'Brooklyn Naval Yard')

plot(0,0,'wx')
text(-4,0, 'lat 40 25')
text(-4, -1, 'lon 74 10')

plot(15, 0, 'wx')
text(15.3,0, 'lat 40 25')
text(15.3, -1, 'lon 73 50 21')

plot(0, 20, 'wx')
text(-4,20, 'lat 40 45')
text(-4,19, 'lon 74 10')

% dummy return
y = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function type = plottype (n) ;
% File Name : plottype.m
% This function assigns a plot symbol and color for a given track.
% max. choice is colour_size * symbol_size

colour = 'rymcgbw' ;
colour_size = 7 ;

symbol = '-:.+o*x' ;
symbol_size = 7 ;
```

```
sym_s = symbol ( floor (n / colour_size) + 1 ) ;
col_s = colour ( rem(n, colour_size) + 1 ) ;

type = [sym_s col_s] ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ship_name = id2ship (ship_id) ;
% File Name : id2ship
% A look up table to look up ship name for a given ship ID.

global shiplist ;

if ship_id == 0
  ship_name = 'Unknown' ;
else
  ship_id = shiplist (ship_id) ;
end

if ship_id == 1
  ship_name = 'Merchant Patriot' ;
  return ;
end

if ship_id == 2
  ship_name = 'Morgan Reinauer' ;
  return ;
end

if ship_id == 3
  ship_name = 'Sea Lion' ;
  return ;
end

if ship_id == 4
  ship_name = 'Steven F O''Hara' ;
  return ;
end

if ship_id == 5
  ship_name = 'Wal-Row' ;
  return ;
end

if ship_id == 6
  ship_name = 'Francis E Roehirg' ;
  return ;
end

if ship_id == 7
  ship_name = 'Cissi Reinauer' ;
```

```
  return ;
 end

 if ship_id == 8
  ship_name = 'Fright' ;
  return ;
 end

 if ship_id == 9
  ship_name = 'Catherine Brown' ;
  return ;
 end

 if ship_id == 10
  ship_name = 'Franklin Reinauer' ;
  return ;
 end

 if ship_id == 11
  ship_name = 'Wal-Row' ;
  return ;
 end

 if ship_id == 12
  ship_name = 'Nan McKay' ;
  return ;
 end

 if ship_id == 13
  ship_name = 'Stephen Reinauer' ;
  return ;
 end

 if ship_id == 14
  ship_name = 'Zim Livorno' ;
  return ;
 end

 if ship_id == 15
  ship_name = 'Terror' ;
  return ;
 end

 if ship_id == 16
  ship_name = 'Newtown Creek' ;
  return ;
 end

 if ship_id == 17
  ship_name = 'Buchanan 10' ;
```

```
  return ;
 end

 if ship_id == 18
  ship_name = 'Eastern Sun' ;
  return ;
 end

 if ship_id == 19
  ship_name = 'Chem Trader' ;
  return ;
 end

 if ship_id == 20
  ship_name = 'Dean Reinauer' ;
  return ;
 end

 if ship_id == 21
  ship_name = 'Buchanan 10' ;
  return ;
 end

 if ship_id == 22
  ship_name = 'Steven F O''hara' ;
  return ;
 end

 if ship_id == 23
  ship_name = 'Itco XII' ;
  return ;
 end

 if ship_id == 24
  ship_name = 'Britannia Mcallister' ;
  return ;
 end

 ship_name = 'unknown' ;
```

Part 4 : DATA CONVERSION AND FORMATTING PROGRAMS

The program files are : course.m, crs_diff.m, radians.m, km2lonla.m, lonlat2k.m, km2nm.m. nm2km.m, num2grid.m, dtg2time, sec2time.m, time2sec.m, timediff.m, inc_sec.m.

```
function crs = course(x,y)
% File Name : course.m
```

```
% This function computes the vessel course in angle.

if x > 0                    % handles quad 1&4, 000, 090, 180
   crs = 90 - atan(y/x)*180/pi;
elseif x < 0                        % handles quad 2&3, 270
   crs = 270 - atan(y/x)*180/pi;
elseif x == 0
  if y > 0
    crs = 0 ;
  elseif y < 0
    crs = 180 ;
  elseif y == 0
    % the ship never moves ==> use previous course
    crs = 999 ;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function crs_ = crs_diff (crs1, crs2) ;
% File Name : crs_diff.m
% This function is used in distance measure computation.

delta = crs2 - crs1 ;

if crs2 >= crs1
  if delta < 360 - delta
    crs_ = delta ;
  else
    crs_ = (-1) * (360 - delta) ;
  end
else
  if abs (delta) < 360 + delta
    crs_ = delta ;
  else
    crs_ = 360 + delta ;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function rad = radians (deg) ;
% File Name : radians.m
% function takes an input in degrees and converts it to radians

rad = deg * pi / 180;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [lon, lat] = km2lonlat(lon_orig,lat_orig,east,north)
% File Name : km2lonla.m
%KM2LONLAT  Convert distances in km referenced to a lon/lat point to lon/lat.
%
```

```
%   This function will convert distances in kilometers east and west
%   of a reference longitude/latitude point to longitude/latitude.  The
%   equation used is from Bowditch's book "The American Practical Navigator."
%
% Usage:
%   [LON,LAT]=KM2LONLAT(LON_ORIG,LAT_ORIG,EAST,NORTH)
%
% Inputs:
%     LON_ORIG - reference longitude.
%     LAT_ORIG - reference latitude.
%     EAST     - distance east (km) of reference point (scalar or vector).
%     NORTH    - distance north (km) of reference point (scalar of vector).
%
% Outputs:
%     LON      - longitude
%     LAT      - latitude
%
% Example:
%               [LON,LAT]=KM2LONLAT(-122,35.4,EAST,NORTH)
%         will convert the vectors EAST and NORTH, which contain distances
%          in km east and north of -122 W, 35.4 N to lon/lat pairs, returned
%         in the vectors LON and LAT.

%        Mike Cook - NPS Oceanography Dept. - FEB 94
%       Mike Cook - JUN 94 - added more documentation and error checking.

% Check for the correct number of inputs.
        if nargin ~= 4
            error(' You *MUST* supply 4 input arguments ')
        end

        con = radians(lat_orig);
        ymetr = 111132.09 - 566.05 .* cos( 2 .* con) ...
             + 1.2 .* cos(4 .* con) - 0.002 .* cos(6 .* con);
        xmetr = 111415.13 .* cos(con) - 94.55 .* cos(3 .* con) ...
             + 0.012 .* cos(5 .* con);
        lon = east .* 1000 ./ xmetr + lon_orig;
        lat = north .* 1000 ./ ymetr + lat_orig;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [east, north] = lonlat2km(lon_orig,lat_orig,lon,lat)
% File Name : lonlat2k.m
%LONLAT2KM  Convert lat/lon to distances (km) referenced to a lon/lat point.
%
%  This function will convert longitude/latitude pairs to distances in
%  kilometers east and west of a reference longitude/latitude point.  The
%  equation used is from Bowditch's book "The American Practical Navigator."
%
% Usage:
```

```
% [EAST,NORTH]=LONLAT2KM(LON_ORIG,LAT_ORIG,LON,LAT)
%
%
% Inputs:  lon_orig - reference longitude.
%          lat_orig - reference latitude.
%          lon      - longitude scalar or vector.
%          lat      - latitude scalar or vector.
%
% Outputs: east     - distance east from reference point (km)
%          north    - distance north from reference point (km)
%
% Example:
%               [EAST,NORTH]=LONLAT2KM(-122,35.4,LON,LAT)
%          will convert the vectors LON and LAT, which contain lon/lat pairs,
%           to distances in km east and north of -122 W, 35.4 N, returned
%          in the vectors EAST and NORTH.

%       Mike Cook - NPS Oceanography Dept. - FEB 94
%      Mike Cook - JUN 94 - added more documentation and error checking.

      if nargin ~= 4
        error(' You *MUST* supply 4 input arguments ')
      end

      con = radians(lat_orig);
      ymetr = 111132.09 - 566.05 .* cos(2 .* con) + 1.2 ...
           .* cos(4 .* con) - 0.002 .* cos(6 .* con);
      xmetr = 111415.13 .* cos(con) - 94.55 .* cos(3 .* con) ...
           + 0.012 .* cos(5 .* con);
    east = (lon - lon_orig) .* xmetr ./ 1000;
    north = (lat - lat_orig) .* ymetr ./ 1000;
```

```
function nm = km2nm (km)
% File Name : km2nm.m
% This function converts km to nautical miles

nm = km ./ 1.852 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function km = nm2km(nm)
% File Name : nm2km.m
% This function converts nautical miles to km.

km = nm * 1.852 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function latlong = num2grid (num) ;
% File Name : num2grid.m
% This function converts lat. or long. in concatenated form - deg|min|sec to
% degrees.

if num < 0
  sign = -1 ;
else
  sign = + 1 ;
end

num = abs (num) ;

deg = floor (num / 10000) ;
leftover = num - deg * 10000 ;
min = floor (leftover / 100) ;
sec = leftover - min * 100 ;

latlong = sign * (deg + min/60 + sec/3600) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function time = dtg2time (dtg) ;
% File Name : dtg2time.m
% This function converts DTG concatenated number to 24-hour unit.

time = dtg - floor (dtg / 10000) * 10000 ;

% addition on 29 Oct 94 to change time to seconds resolution for fine fusion.
time = time * 100 ;
% actually not necessary here

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hhmmss = sec2time (hh, mm, ss) ;
% File Name : sec2time.m
% This function converts seconds to time format.
```

```
hhmmss = hh * 10000 + mm * 100 + ss ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [hr, min, sec] = time2sec (hr_min_sec) ;
% File Name : time2sec.m
% This function converts 24-hr clock concatenated number to hour, min, sec.

hr = floor (hr_min_sec / 10000) ;
min_sec = hr_min_sec - hr * 10000 ;

min = floor (min_sec / 100) ;
sec = min_sec - min * 100 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function time_sec = timediff (hhmmss1, hhmmss2) ;
% File Name : timediff.m
% This function computes the time difference in seconds between 2 given time.

[hh, mm, ss] = time2sec (hhmmss1) ;
sec1 = hh * 3600 + mm * 60 + ss ;

[hh, mm, ss] = time2sec (hhmmss2) ;
sec2 = hh * 3600 + mm * 60 + ss ;

time_sec = sec2 - sec1 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function time = inc_sec (hhmmss, inc) ;
% File Name : inc_sec.m
% This function increment a given time by inc seconds.

[hh, mm, ss] = time2sec (hhmmss) ;

ss = ss + inc ;
carry_min = floor (ss / 60) ;
if carry_min > 0
  ss = ss - 60 ;
end

mm = mm + carry_min ;
carry_hr = floor (mm / 60) ;
if carry_hr > 0
  mm = mm - 60 ;
end

hh = rem (hh + carry_hr, 24) ;

time = sec2time (hh, mm, ss) ;
```

LIST OF REFERENCES

1. Hall, D. L., *Mathematical Techniques in Multisensor Data Fusion*, Artech House, Norwood, Massachusetts, 1992.

2. Haykin, S., *NEURAL NETWORKS: A Comprehensive Foundation*, Macmillan College Publishing Company, Inc., New York, New York, 1994.

3. Hartline, P. H., Glass, L., and Loop, M. S., "Merging of modalities in the optic tectum: infrared and visual integration in rattlesnake," *Science*, Vol. 199, pp. 1225-1229, 1978.

4. Ajjimarangsee, P. and Huntsberger, T. L., "Neural Network Model for Fusion of Visible and Infrared Sensor Outputs," *Proc. SPIE - Sensor Fusion: Spatial Reasoning and Scene Interpretation*, Vol. 1003, pp. 153-160, Cambridge, Massachusetts, 1988.

5. Pearson, J. C., et al., "Neural network approach to sensory fusion," *Proc. SPIE - Sensor Fusion*, Vol. 931, pp. 103-108, Orlando, Florida, 1988.

6. Levine, R. Y. and Khuon, T. S., "Neural Network for Distributed Sensor Data Fusion: The Firefly Experiment," *Proc. SPIE - Sensor Fusion IV*, Vol. 1611, pp. 52-63, Boston, Massachusetts, 1991.

7. Brown, D. E., et al., "Neural Network Implementations of Data Association Algorithms for Sensor Fusion," *Proc. SPIE - Sensor Fusion II*, Vol. 1100, pp. 126-135, Orlando, Florida, 1989.

8. Kagel, J. H., et al., "Multispectral image fusion using neural network," *Proc. SPIE - Applications of Artificial Neural Network*, Vol. 1294, pp. 180-186, Orlando, Florida, 1990.

9. Waltz, E. L., "Data Fusion for C3I: A Tutorial," *Command, Control, Communications Intelligence (C3I) Handbook*, EW Communications, Inc., pp. 217-226, Palo Alto, California, 1986.

10. Nahin, P. J. and Pokoski, J. L., "NCTR Plus Sensor Fusion Equals IFFN," *Transactions on Aerospace Electronic Systems*, Vol. AES-16, No. 3, pp. 320-327, May 1980.

11. *Resolution A578/14 of the International Maritime Organization: Guidelines for Vessel Traffic Services*, International Maritime Organization, London, 1985.

12. Young, W., "What are Vessel Traffic Services, and What Can They Really Do ?," *Navigation*, Vol. 41, No. 1, pp. 31-55, Spring 1994.

13. System Design Document for the Coast Guard Vessel Traffic Service System, Aircraft Division, Naval Air Warfare Center, Patuxent River, Maryland, March 1994.

14. Alsip, D. H., Butler, J. M., and Radice, J. T., "The Coast Guard's Differential GPS Program," *Navigation*, Vol. 39, No. 4, pp. 345-361, Winter 1992-93.

15. Kohonen, T., "The self-organizing map," *Proceedings of the IEEE*, Vol. 78, pp. 1464-1480, 1990.

16. Bowditch, N., *American Practical Navigator: An Epitome of Navigation*, Defence Mapping Agency Hydrographic/Topographic Center, Washington, D.C., 1984.

17. Grossberg, S. and Carpenter, G. A., "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics and Image Processing*, Vol. 37, pp. 54-115, 1987.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5101	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
4.	Professor Murali Tummala, Code EC/Tu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	5
5.	Professor Gurnam S. Gill, Code EC/Gl Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
6.	LCDR Michael Linzey P. O. Box 60 USCG EECEN Wildwood Crest, New Jersey 08260-0060	1
7.	LT John Wood P. O. Box 60 USCG EECEN Wildwood Crest, New Jersey 08260-0060	1
8.	Chief Defence Scientist MINDEF Singapore MINDEF Building, Gombak Drive, S 2366 Republic of Singapore	1

9. Director 1
Defence Science Organisation
20 Science Park Drive, S 0511
Republic of Singapore

10. Divisional Manager, Radar Research Division 1
Defence Science Organisation
20 Science Park Drive, S 0511
Republic of Singapore

11. Leonard P. Koh 2
17-R, Jalan Hock Chye, S 1953
Republic of Singapore